



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BARCELONATECH

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Automatic differentiation in probabilistic machine learning with variational inference

Alberto Pou Quirós

Máster en Ingeniería Informática

TUTORES: Joan Capdevila Pujol y Jordi Torres Viñals

6 de julio de 2017

Abstract

La era tecnológica que se vive actualmente está dando lugar a cantidades ingentes de información que requieren ser analizadas de forma automática. El *machine learning* puede ayudar a mitigar esta sobrecarga encontrando patrones y regularidades en los datos, los cuales pueden contener información relevante para realizar una determinada tarea. Áreas como la robótica o la medicina utilizan algoritmos de *machine learning* para decidir el siguiente movimiento que debe hacer un robot o para diagnosticar una posible enfermedad dados unos síntomas. Durante el último año, el mejor jugador de *Go* ha sido derrotado dos veces por un algoritmo de *Google* basado en *deep learning* y *reinforcement learning*, su traductor está mejorando cada día y su aplicación de fotos ya detecta e identifica las caras de tus amigos y los lugares donde te haces los *selfies*. Todo esto sucede mientras *Amazon* te aconseja qué productos deberías comprar y *Tesla* ya comercializa sus coches autónomos.

El *probabilistic machine learning* se basa en el uso de las reglas de la teoría de la probabilidad para conseguir que las máquinas aprendan a identificar patrones en los datos. La utilización de la teoría de la probabilidad permite a estos métodos modelar la incertidumbre, un aspecto muy importante a la hora de tomar decisiones. Este tipo de *machine learning* permite separar la parte de modelización de la parte de inferencia. De esta forma, se pueden aplicar diversos algoritmos de inferencia independientemente del modelo probabilístico.

Esta tesis de máster da una visión global del estado del arte en los *frameworks* de *probabilistic programming*, centra el análisis en torno a la aproximación variacional de *Bayesian Inference* e investiga el uso de *Automatic Differentiation* (AD) para automatizar ésta. Este documento se ha escrito de forma autocontenida con el objetivo de que el lector sepa exactamente de dónde provienen cada una de las fórmulas y algoritmos que aquí se explican. Para ello se ha seleccionado un modelo concreto, *Gaussian Mixture Model* (GMM), y se estudia el uso de la AD, a través de la librería *Tensorflow*, para su inferencia mediante diferentes métodos de *Variational Inference* (VI).

Nowadays the technology age that we are living in is generating massive amounts of data which requires automatic processing. Machine learning can help in mitigating this overload by finding patterns or regularities on data which contain relevant information for a specific task. Areas like robotics or medicine are using machine learning algorithms to decide the next move of a robot or to diagnose possible diseases given some symptoms. During the last year, the best Go player has been defeated twice by a Google artificial intelligence based on deep learning and reinforcement learning, their translator is improving every day, and their photo application already detects and identifies your friend's faces and the places where you are doing selfies. All this happens while Amazon decides what products you should buy and Tesla markets its autonomous cars.

Probabilistic machine learning is based on the use of the probability theory rules so computers can identify patterns in data. Using probability theory allows these models to deal with uncertainty, a key aspect in decision-making. This type of machine learning separates the modeling process of the inference process. In this way, several inference algorithms can be applied in the same probabilistic model.

This master thesis gives an overview of probabilistic programming frameworks. It focuses on the analysis in the variational approximation of Bayesian Inference and studies the use of Automatic Differentiation (AD) to automate it. This document has been written in a self-contained way to let the reader know exactly where each formula and each algorithm comes from. For this, a particular probabilistic model was chosen, Gaussian Mixture Model (GMM), with the aim of studying the use of AD in it, with Tensorflow library and using different Variational Inference (VI) methods.

Agradecimientos

En primer lugar, me gustaría agradecer a Joan Capdevila Pujol por introducirme al *probabilistic machine learning* y por ayudarme a entender todas las derivaciones necesarias para la realización de este proyecto. También agradecer a Jordi Torres por darme la oportunidad de trabajar en el *Barcelona Supercomputing Center* (BSC) y aprender todo lo que he aprendido aquí.

En segundo lugar, quiero agradecer a Guillem, Cesare, Fabriccio, Eloy, Pol, Maurici, Miriam y Víctor del *Autonomic Systems and e-Business Platforms group* del BSC porque, aún siendo del Real Madrid y apoyar a Jorge Lorenzo, han hecho que mi estancia en el BSC sea muy divertida.

En tercer lugar, tengo que agradecer a la familia que me ha aguantado este último año en Barcelona, mis compañeros de piso Xisco, Claus y Carlos. Todo ha sido mucho más divertido gracias a vuestra compañía. Me costará ver *First Dates* sin comentarlo con vosotros y echaré mucho de menos nuestras partidas al *Mario Kart*. Pero sobretodo tengo que agradecerse a Xisco, mi buen amigo de la carrera y con el que he tenido la suerte de coincidir en Barcelona. Gracias a su ayuda incondicional siempre que la he necesitado, tanto este proyecto como el máster han salido adelante más fácilmente.

Finalmente agradecer a mi familia, que tanto desde Mallorca como desde Cambrils, me han apoyado siempre.

Índice

Lista de acrónimos	7
1 Introducción	9
1.1 ¿En qué consiste el machine learning?	9
1.2 ¿Qué es el probabilistic machine learning?	10
1.3 Motivación	11
1.4 Objetivos del proyecto	12
1.5 Estructuración del documento	13
2 Automatic Differentiation	15
2.1 Tipos de derivación	15
2.2 Automatic Differentiation forward mode	16
2.3 Automatic Differentiation reverse mode	17
2.4 Herramientas de derivación	17
2.5 Tensorflow	18
2.6 Ejemplo de uso	19
3 Variational Inference	21
3.1 Probabilistic machine learning	21
3.2 Bayesian Inference	22
3.2.1 Regla de Bayes	22
3.2.2 Aproximación del posterior	23
3.2.3 Probabilistic graphical models	24
3.3 Variational Inference	25
3.3.1 Estrategia	25
3.3.2 Procedimiento	27
3.3.3 Algoritmo	28
3.4 Coordinate Ascent Variational Inference	28
3.4.1 Derivación genérica	28
3.4.2 Derivación mediante las propiedades de la Exponential Family	29
3.4.3 Algoritmo	31
3.4.4 Ejemplo	31
3.5 Gradient Ascent Variational Inference	32
3.5.1 Gradient ascent	32
3.5.2 Algoritmo	32
3.5.3 Ejemplo	32
3.5.4 Consideraciones	33
3.6 Problemas de eficiencia	33
3.7 Stochastic Variational Inference	33
3.7.1 Stochastic optimization	34
3.7.2 Algoritmo	34
3.8 Black Box Variational Inference	35
3.8.1 Score gradients	35
3.8.2 Monte-Carlo integration	35
3.8.3 Algoritmo	35
3.8.4 Consideraciones	36
3.9 Librerías de Variational Inference	36
4 Gaussian Mixture Model	37
4.1 Proceso generativo	37

4.2	Probabilistic graphical model y probabilidad conjunta	37
4.3	Intratabilidad del posterior	38
4.4	Modelo variacional	38
4.5	Derivación de los parámetros variacionales	39
4.6	Model Evidence Lower Bound (ELBO)	40
4.7	Implementaciones	40
4.7.1	Coordinate Ascent Variational Inference	40
4.7.2	Gradient Ascent Variational Inference	41
4.7.3	Stochastic Coordinate Ascent Variational Inference	42
4.7.4	Stochastic Gradient Ascent Variational Inference	42
4.8	Otros modelos implementados	42
5	Experimentación	43
5.1	Comparación de ELBOs	43
5.2	Comparación del consumo de memoria	44
5.3	Comparación de tiempos	44
5.4	Comparación entre optimizadores	45
5.5	Mejoras por hardware	46
6	Aplicaciones	49
6.1	Datasets	49
6.1.1	Datasets sintéticos	49
6.1.2	Dataset de Mallorca	49
6.1.3	Dataset de Porto	49
6.2	Preprocessing	50
6.2.1	Interpolación de los datos	50
6.2.2	Reducción de dimensionalidad	51
6.3	Resultados	53
6.3.1	Resultados con datasets sintéticos	53
6.3.2	Resultados con el dataset de Mallorca	54
6.3.3	Resultados con el dataset de Porto	58
6.4	Generación de nuevos datos	58
7	Conclusiones	63
7.1	Lecciones aprendidas	63
7.2	¿Qué he aprendido?	64
7.3	Dificultades	64
7.4	Futuros pasos	65
A	Apéndices	67
A.1	Apéndices del capítulo 2	67
A.1.1	Derivación completa de la evidencia para Variational Inference (VI)	67
A.1.2	Reescritura de la ELBO	67
A.1.3	Derivación genérica completa de un parámetro variacional	67
A.1.4	Aproximación de una distribución variacional suponiendo mean-field assumption	68
A.1.5	Formulación de una distribución Dirichlet en Exponential Family	68
A.1.6	Derivación score gradients de la ELBO	69
A.2	Apéndices del capítulo 4	70
A.2.1	Distribuciones del modelo GMM formuladas en términos de la Exponential Family	70
A.2.2	Cálculo de expectations	71
A.2.3	Formulación del parámetro variacional λ_π en términos de la Exponential Family	74
A.2.4	Formulación de los parámetros variacionales λ_m , λ_β , λ_ν y λ_W en términos de la Exponential Family	75
A.2.5	Formulación del parámetro variacional λ_ϕ en términos de la Exponential Family	76
A.2.6	Derivación completa de la ELBO	78
	Referencias	81

Lista de acrónimos

AD	<i>Automatic Differentiation</i>
ADVI	<i>Automatic Differentiation Variational Inference</i>
ARD	<i>Automatic Relevance Determination</i>
BBVI	<i>Black Box Variational Inference</i>
BSC	<i>Barcelona Supercomputing Center</i>
CAVI	<i>Coordinate Ascent Variational Inference</i>
CNN	<i>Convolutional Neural Networks</i>
CPU	<i>Central Processing Unit</i>
CUDA	<i>Compute Unified Device Architecture</i>
DAG	<i>Directed Acyclic Graph</i>
DNN	<i>Deep Neural Networks</i>
DDoS	<i>Distributed Denial of Service</i>
ELBO	<i>Evidence Lower Bound</i>
GAVI	<i>Gradient Ascent Variational Inference</i>
GMM	<i>Gaussian Mixture Model</i>
GPS	<i>Global Positioning System</i>
GPU	<i>Graphic Processing Unit</i>
GPX	<i>GPS Exchange Format</i>
HIPS	<i>Harvard Intelligent Probabilistic Systems Group</i>
HMC	<i>Hamiltonian Monte Carlo</i>
KL	<i>Kullback-Leibler divergence</i>
MAP	<i>Maximum A Posteriori estimation</i>
MCMC	<i>Markov Chain Monte Carlo</i>
MFVI	<i>Mean-Field Variational Inference</i>
MLE	<i>Maximum Likelihood Estimation</i>
MSE	<i>Mean Squared Error</i>
NaN	<i>Not a Number</i>
NIPS	<i>Neural Information Processing Systems</i>
PCA	<i>Principal Component Analysis</i>
PPCA	<i>Probabilistic Principal Component Analysis</i>
PDF	<i>Probability Density Function</i>
RNN	<i>Recurrent Neural Networks</i>
SCAVI	<i>Stochastic Coordinate Ascent Variational Inference</i>
SGAVI	<i>Stochastic Gradient Ascent Variational Inference</i>
SNGAVI	<i>Stochastic Natural Gradient Ascent Variational Inference</i>
SVI	<i>Stochastic Variational Inference</i>
TFG	<i>Trabajo de Fin de Grado</i>
VI	<i>Variational Inference</i>

1. Introducción

En este primer capítulo se explica de forma general qué es el *machine learning* y cuáles son las principales características de su vertiente probabilística: el *probabilistic machine learning*. También se comentan los motivos e ideas principales que nos llevaron al desarrollo de este proyecto.

1.1 ¿En qué consiste el machine learning?

Actualmente, en la era del *big data*, se dispone de cantidades masivas de información que requiere ser procesada de forma automática. Se conoce como *machine learning* al conjunto de técnicas existentes para la detección de patrones en los datos, para luego utilizarlos en tareas como la predicción, la toma de decisiones, la clasificación o en la generación de nuevos datos. Una gran parte de los algoritmos de *machine learning* optimizan una función objetivo. Para ello, construyen un modelo, una representación compacta de los datos, que permitirá realizar predicciones sobre futuros ejemplos. Esto pasa por optimizar una serie de parámetros con el objetivo de minimizar o maximizar una función. Los gradientes, la generalización multivariada de la derivada, son el pilar sobre el que se sostienen este tipo de optimizaciones. El gradiente nos indica la dirección de máxima pendiente de la función en un punto. Y es lo que se utiliza, de forma iterativa, para desplazarse por el espacio de la función y encontrar mínimos y máximos relativos.

Dada la importancia de esta operación matemática los investigadores han desarrollado diferentes metodologías con el objetivo de calcular estos gradientes de la forma más eficiente posible. *Numerical Differentiation*, *Symbolic Differentiation* y *Automatic Differentiation* (AD) son algunos ejemplos. De este problema nacen paquetes *software* para el cálculo de gradientes como *Tensorflow* [1], *Autograd* [2], *Theano* [3] o *Torch* [4]. Estas librerías utilizan alguna de las metodologías de derivación comentadas anteriormente para permitir a los desarrolladores optimizar sus modelos. En este proyecto utilizaremos AD para la optimización del modelo y mediante un paquete que incluye su implementación, *Tensorflow*, se comprobará su eficacia para la inferencia en modelos probabilísticos.

Se distinguen dos grandes formas de aprendizaje de modelos en el *machine learning*: el aprendizaje supervisado y el no supervisado. Por un lado, cuando el objetivo es predecir el valor de una variable, se suele optar por el aprendizaje supervisado. Este tipo de aprendizaje requiere de un conjunto de datos de entrenamiento y un conjunto de test. Cada ejemplo del conjunto de entrenamiento viene acompañado del resultado de la variable a predecir, de forma que el objetivo de este tipo de aprendizaje pasa por entrenar un modelo, modelar una función, que dadas las características del ejemplo (*features*), obtenga la salida deseada de la variable a predecir (*target*). Después se utiliza el conjunto de test para verificar lo bien que generaliza el modelo entrenado con la intención de evitar que haya memorizado el conjunto de entrenamiento (*overfitting*), en vez de haber capturado las características esenciales del conjunto de datos. Algunos algoritmos de este tipo de aprendizaje son la regresión lineal [5], las redes neuronales [6] o los árboles de decisión [7]. Por otro lado, cuando lo que se requiere es describir unos datos, se suelen utilizar algoritmos para el aprendizaje no supervisado. En contraposición al aprendizaje supervisado, éste solo necesita un conjunto de datos. La meta de este tipo de algoritmos es obtener agrupaciones en los datos (*clustering*) para identificar y caracterizar a los individuos de la población. Algunos algoritmos de este tipo de aprendizaje son *KMeans* [8], *DBSCAN* [9] o *Gaussian Mixture Model* (GMM) [10].

En este proyecto se estudia el modelo GMM. Este modelo de aprendizaje no supervisado es probabilístico y se puede representar mediante un *probabilistic graphical model* [11, 12]. El aprendizaje de sus parámetros dados unos datos, la inferencia, se puede realizar por *Maximum Likelihood Estimation* (MLE) [13], *Maximum A Posteriori estimation* (MAP) [14] o *Bayesian Inference* [15]. En esta tesis nos centramos en la aproximación variacional de *Bayesian Inference*.

1.2 ¿Qué es el probabilistic machine learning?

El *probabilistic machine learning* utiliza la teoría de la probabilidad [16] para resolver los problemas del *machine learning*. La utilización de la teoría de la probabilidad permite a estos métodos modelar la incertidumbre, un aspecto muy importante a la hora de tomar decisiones. Esta aproximación probabilística obtiene modelos dependientes de los datos que permiten saber ¿cuál es la mejor predicción o decisión dados unos datos? o ¿cuál es el mejor modelo dados unos datos?. Como ya se ha comentado, este tipo de *machine learning*, separa la parte de modelización de la parte de inferencia del modelo. De esta forma, se puede definir un modelo probabilístico para después elegir el método de inferencia de manera independiente.

La construcción de este tipo de modelos se hace siguiendo la filosofía del *Box Loop*. El *Box Loop* es un esquema, creado por el estadístico George E. P. Box¹, sobre el cual se itera repetidas veces durante la construcción de un modelo probabilístico [17, 18].

1. Primero se construye el modelo en base a los conocimientos que se tienen sobre ese entorno.
2. En segundo lugar, se descubren patrones en los datos utilizando el modelo definido previamente y un método de inferencia.
3. Finalmente se valida el modelo. Si no fuera lo suficientemente bueno se volvería al paso 1 y en caso contrario, se utilizaría para describir o predecir nuevos datos.

En la figura 1.1 se muestra un esquema gráfico del *Box Loop* comentado. La modelización consiste en definir el

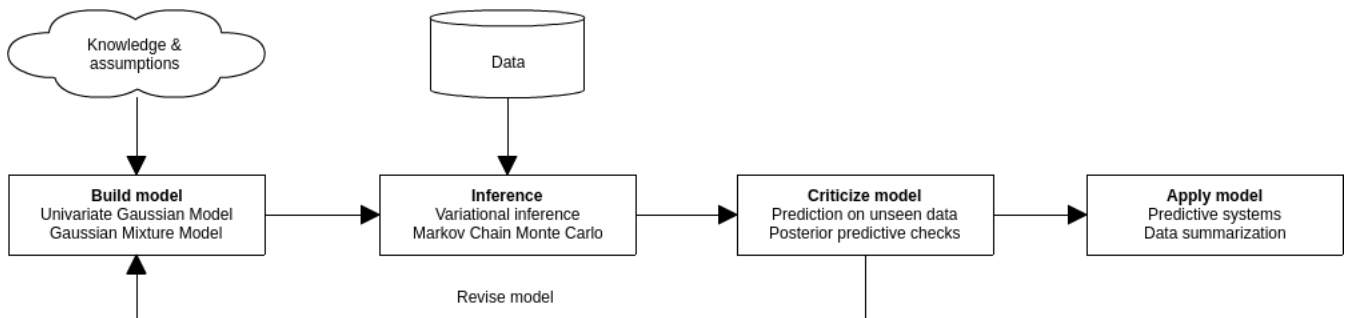


Figura 1.1: *Box Loop*

mejor modelo que haya podido generar los datos disponibles. Esto empieza por definir las diferentes variables del problema y sus relaciones de dependencia, luego se utilizan distribuciones de probabilidad para definir la forma y relaciones de cada una de ellas y finalmente se mapean algunas con los datos disponibles. Normalmente esta definición requiere de la colaboración con un experto. Para representar y estructurar los modelos probabilísticos se utilizan los *probabilistic graphical models*. Dependiendo de si las relaciones entre las variables del modelo son en un sentido o en los dos se conocen de dos tipos:

- *Markov networks*. Cuando la relación entre cada par de variables es bidireccional. Da lugar a un grafo no dirigido. Como por ejemplo el *Ising model* [19].
- *Bayesian networks*. Cuando la relación entre cada par de variables es unidireccional (probabilidades condicionales). Da lugar a un grafo dirigido. Como por ejemplo el GMM.

En este proyecto se trabaja con este segundo tipo.

El proceso de inferencia de un modelo probabilístico consiste en revertir su proceso generativo. Esto es, dados unos datos, inferir los parámetros del modelo que mejor explican esos datos. Para este fin existen aproximaciones como MLE, MAP o *Bayesian Inference*. *Bayesian Inference* consiste en aproximar la distribución *posterior* asumiendo unos *priors* sobre cada una de las variables aleatorias que conforman el modelo. Existen dos ramas algorítmicas para la aproximación de la distribución *posterior*:

1. *Markov Chain Monte Carlo* (MCMC) [20]: *Sampling approximate inference*.
2. *Variational Inference* (VI) [21]: *Structural approximate inference*.

El proceso de inferencia que se utilizará será VI [22]. Se trata de un tipo de *Bayesian Inference* cuyo objetivo es optimizar un conjunto de parámetros para aproximar la distribución *posterior*. Como se verá, la problemática

¹George E. P. Box: https://en.wikipedia.org/wiki/George_E._P._Box

aparece cuando se intenta modelar dicha distribución en espacios de muchas dimensiones o en modelos no conjugados, donde las soluciones analíticas pasan a ser intratables.

Un aspecto importante para realizar la modelización es el lenguaje a utilizar. Actualmente existen numerosas librerías que ofrecen un lenguaje para la modelización de este tipo de modelos. Es lo que se conoce como *probabilistic programming*. Estas herramientas permiten definir un modelo probabilístico y después elegir entre diferentes algoritmos para la inferencia. Las principales diferencias entre estas librerías son:

- Lenguaje de programación.
- Tipos de variables permitidas en la inferencia (variables categóricas, numéricas o ambas).
- Métodos de inferencia disponibles.
- Definición de modelos dirigidos y no dirigidos.
- Distribución en diferentes *Central Processing Unit* (CPU) o *Graphic Processing Unit* (GPU).

En la siguiente tabla² se muestran algunos de los paquetes *software* disponibles más conocidos.

Nombre	Tipos de inferencia	Lenguaje	GPU
BUGS	<i>Gibbs sampling</i> [23], <i>Metropolis-Hastings</i> [24], ...	Propio	No
<i>Figaro</i>	<i>Gibbs sampling</i> , <i>Metropolis-Hastings</i> , <i>exact inference</i> , MAP, ...	<i>Scala</i>	No
<i>Nimble</i>	<i>Monte Carlo methods</i>	<i>R</i>	No
<i>Infer.net</i>	<i>Expectation propagation</i> [25], <i>Gibbs sampling</i> , <i>Variational Message Passing</i> [26], ...	<i>C#</i>	No
<i>PyMC</i>	<i>Automatic Differentiation Variational Inference</i> (ADVI), <i>Operator Variational Inference</i> [27], <i>Sequential Monte Carlo</i> [28], <i>Metropolis-Hastings</i> , ...	<i>Python</i>	No
<i>Probabilistic C</i>	<i>Sequential Monte Carlo</i> y <i>Particle MCMC</i>	<i>C</i>	No
<i>Stan</i>	<i>Hamiltonian Monte Carlo</i> (HMC) [29], ADVI, ...	Propio	No
<i>Edward</i> [30]	<i>Gibbs sampling</i> , <i>Metropolis-Hastings</i> , ADVI, <i>Black Box Variational Inference</i> (BBVI), <i>Stochastic Variational Inference</i> (SVI) [31], ...	<i>Python</i>	Sí

Dada la aceptación que está recibiendo y su soporte para GPUs se decidió empezar probando la librería *Edward* ya que además, el *Barcelona Supercomputing Center* (BSC), dispone de un *cluster* de GPUs de *Nvidia* que nos permitiría escalar las diferentes implementaciones. No obstante, se encontraron diversos problemas con esta librería.

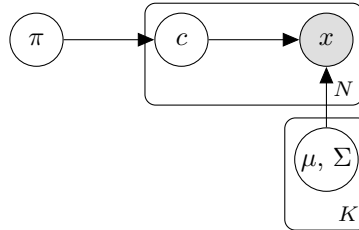
1.3 Motivación

Como se ha dicho, VI se basa en la optimización de una serie de parámetros para la aproximación de la distribución *posterior*. En esta optimización es donde entrará en acción AD. Uno de los objetivos del proyecto es estudiar el comportamiento de este tipo de optimización con diferentes métodos de VI y en diferentes modelos probabilísticos. En definitiva, ver cuando es más aconsejable utilizar AD para optimizar los parámetros para la aproximación del *posterior* de un modelo y cuando no lo es. Hay que tener en cuenta que la derivación analítica de la función objetivo es la forma correcta de proceder para aquellos modelos para los que esta derivación sea tratable. Sin embargo, hay modelos para los que no se puede o no es trivial realizar esta derivación y el uso de AD puede ser de gran ayuda. No obstante, para estas optimizaciones analíticas se requiere de unos conocimientos amplios en matemáticas y concretamente en estadística. Otros grupos han propuesto nuevos esquemas de inferencia que combinan AD con VI [32]. A diferencia de ellos, nosotros tomamos un enfoque más experimental, mostraremos implementaciones que utilizarán AD directamente sobre la función objetivo de VI para un modelo concreto.

Una de las cosas que nos interesó de *Edward* es el uso de la AD de *Tensorflow* para la inferencia de modelos probabilísticos mediante el algoritmo de BBVI [33]. Uno de los ejemplos que se podía encontrar en el repositorio

²Librerías de Bayesian Inference: <http://probabilistic-programming.org/wiki/Home>

de *Edward* era un GMM para dos *clusters*³. Pero modificando los hiperparámetros te dabas cuenta de que no obtenía unos resultados coherentes⁴. El proyecto empezó por investigar como funcionaba *Edward* exactamente y en ver por qué no obtenía los resultados que cabía esperarse para este modelo. Por esta razón, el modelo elegido para el estudio fue GMM. GMM es un modelo de aprendizaje no supervisado de *clustering*. A continuación se muestra su *probabilistic graphical model* de forma simplificada para introducir al lector, en el capítulo 4 se mostrará la representación completa (con los *priors*) y se explicará en detalle. μ y Σ son los parámetros de una distribución *Normal*.



El modelo GMM asume que cada dato es generado por una de las K distribuciones normales. Así que para cada dato, primero se decide qué distribución lo ha generado y después se genera el dato. Para modelar cada una de las variables aleatorias del modelo se utilizaron las siguientes distribuciones:

- π : Una distribución *Dirichlet*⁵. Modela la proporción de asignaciones de puntos a cada uno de los *clusters*.
- c : N distribuciones *Categorical*⁶. Modelan la asignación de cada punto a cada *cluster*.
- μ y Σ : K distribuciones *Normal Inverse Wishart*⁷. Esta distribución modela la media y la matriz de covarianzas de cada uno de los *clusters*.
- x : N distribuciones normales⁸. Modelan la posición de cada punto en base a la media y varianza del *cluster* al que pertenecen.

En las figuras 1.2a y 1.2b se muestran los resultados obtenidos por el modelo GMM para dos ejecuciones con $K = 2$ y $K = 8$ respectivamente. Se puede apreciar como ha modelado las 2 y 8 distribuciones normales multivariadas, es decir, las medias y las matrices de covarianzas de cada *cluster*.

1.4 Objetivos del proyecto

Los objetivos establecidos para este proyecto fueron los siguientes:

- Entender el funcionamiento de VI y sus diferentes variantes.
- Implementar un GMM e inferirlo analíticamente y con métodos basados en gradientes.
- Estudiar el uso de AD en modelos probabilísticos.
- Comparar la inferencia analítica con la inferencia basada en gradientes.
- Implementar las versiones estocásticas de los algoritmos anteriores.
- Aplicar este modelo a un caso de estudio concreto.
- Estudiar el escalado de VI basado en gradientes utilizando GPUs en el entorno de *Tensorflow*.

³GMM: https://github.com/blei-lab/edward/blob/5d41ccb31e3a532d04dc4cdf0667b2245cc050/examples/tf_mixture_gaussian.py

⁴Ejemplo error: <https://github.com/blei-lab/edward/issues/477>

⁵Distribución *Dirichlet*: https://en.wikipedia.org/wiki/Dirichlet_distribution

⁶Distribución *Categorical*: https://en.wikipedia.org/wiki/Categorical_distribution

⁷Distribución *Normal Inverse Wishart*: https://en.wikipedia.org/wiki/Normal-inverse-Wishart_distribution

⁸Distribución *Normal*: https://en.wikipedia.org/wiki/Normal_distribution

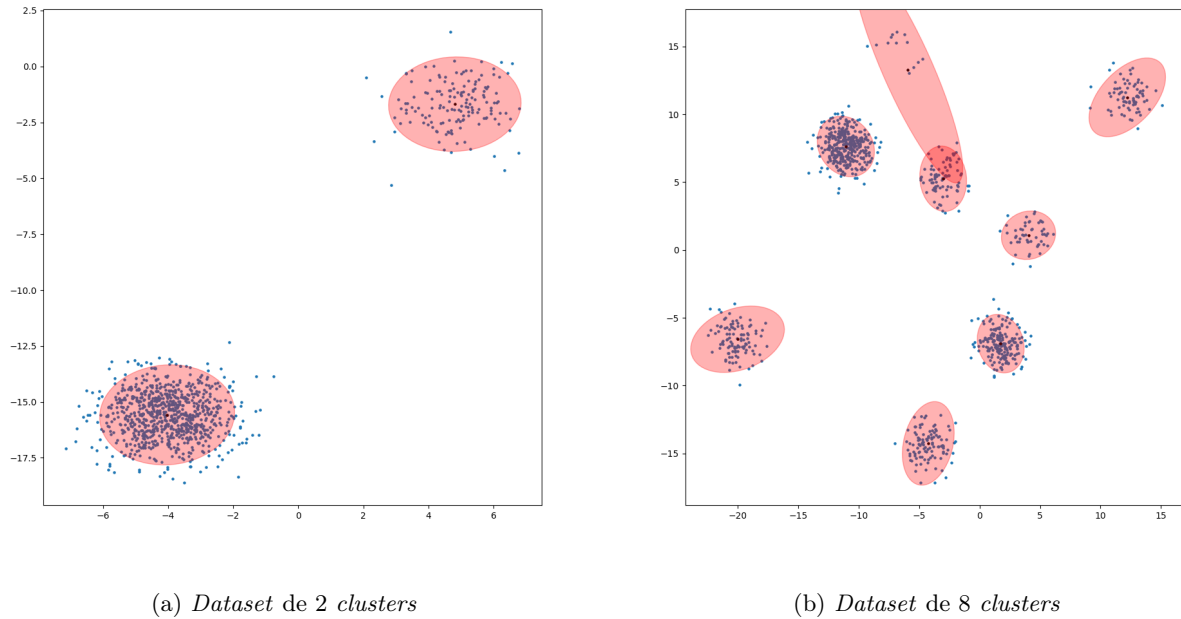


Figura 1.2: Resultados del algoritmo con *datasets* sintéticos de 1000 puntos de 2 dimensiones

1.5 Estructuración del documento

La documentación está estructurada en los siguiente capítulos:

- **Automatic Differentiation.** En este capítulo se presentan los diferentes tipos de derivación que existen y concretamente se explica el método de AD.
- **Variational Inference.** El cometido de este capítulo es poner al lector al día en el estado del arte de los métodos variacionales explicando los términos que se van a utilizar, implementar y discutir a lo largo de todo el proyecto. Con ello se intenta justificar cada uno de los pasos que se han dado en los últimos años en la inferencia de modelos probabilísticos.
- **Gaussian Mixture Model.** En este capítulo se explicará el modelo probabilístico con el que se ha trabajado principalmente en este proyecto: GMM. Se trata de un modelo de aprendizaje no supervisado. Este tipo de aprendizaje procura obtener una descripción de los datos. Concretamente, este modelo agrupa los datos modelando una serie de distribuciones que explicarán su generación. Cuando se habla de *mixtura* se hace referencia a un conjunto de distribuciones del mismo tipo que modelan subconjuntos de datos. En nuestro caso, se modela una *mixtura* de Gaussianas, es decir, una *mixtura* de distribuciones normales.
- **Experimentación.** En este capítulo se presentan diversas comparativas entre los diferentes métodos de VI implementados. Se comparan las versiones basadas en formulación analítica: *Coordinate Ascent Variational Inference* (CAVI) y *Stochastic Coordinate Ascent Variational Inference* (SCAVI); frente a las basadas en gradientes: *Gradient Ascent Variational Inference* (GAVI) y *Stochastic Gradient Ascent Variational Inference* (SGAVI). También se verán las ventajas que ofrecen las versiones estocásticas con respecto a las clásicas y viceversa.
- **Aplicaciones.** En este capítulo se mostrarán los resultados del modelo GMM inferido con diferentes *datasets* (tanto sintéticos como reales) utilizando CAVI.
- **Conclusiones.** En este capítulo se comentan las conclusiones a las que se han llegado durante el desarrollo del proyecto y las líneas abiertas que existen actualmente en estos temas.

2. Automatic Differentiation

En este capítulo se presentan los diferentes tipos de derivación que existen y concretamente se explica el método de *Automatic Differentiation* (AD).

2.1 Tipos de derivación

Los gradientes (la generalización multivariada de la derivada) y las hesianas (matriz cuadrada de las segundas derivadas parciales), se han convertido en un pilar fundamental del *machine learning*. El gradiente es un vector que nos indica la dirección de máxima pendiente de una función en el punto evaluado. Esto es importante para movernos por el espacio de la función y encontrar mínimos y máximos relativos (minimizar y maximizar la función respectivamente). La hesiana nos da información sobre la concavidad y convexidad de la función. Algunos algoritmos la utilizan para mejorar el movimiento exploratorio por el espacio de la función y encontrar los mínimos o máximos locales de forma más óptima. AD es una forma de calcular gradientes de funciones numéricas, representadas como programas informáticos, de forma eficiente y exacta. Como ya se ha comentado el uso de gradientes es muy importante para la optimización de modelos. Desde un punto de vista muy simplista, un modelo se puede entender como una función que se quiere minimizar o maximizar y los gradientes son la herramienta para conseguirlo.

Existen diversos métodos para calcular derivadas con un ordenador [34]:

- *Numerical differentiation*. Este método utiliza la definición de derivada para realizar una aproximación mediante muestras de la función original. De esta manera podemos aproximar el gradiente

$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$ como:

$$\frac{\partial f(x)}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(x + he_i) - f(x)}{h}$$

donde e_i es el i -ésimo vector unitario (sirve para seleccionar la componente del espacio) y $h > 0$ es el

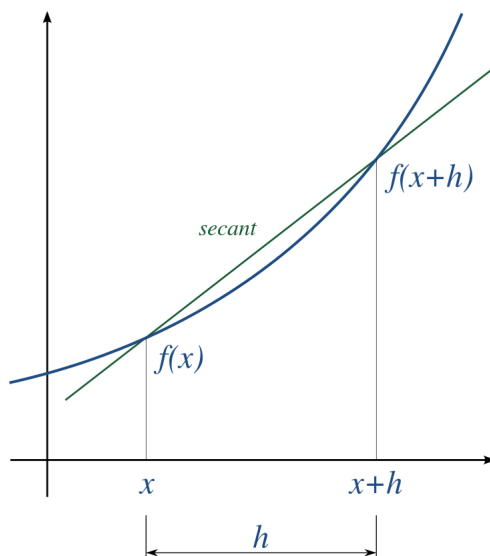


Figura 2.1: *Numerical Differentiation* (Fuente: Wikipedia)

tamaño del paso para la aproximación. Requiere n evaluaciones para obtener el gradiente en un espacio de n dimensiones.

- *Symbolic differentiation*. Consiste en la manipulación automática de las expresiones para la obtención de las derivadas. Requiere de la implementación de las reglas de derivación. El problema de este tipo de derivación es que nos puede llevar a largas expresiones simbólicas difíciles de evaluar.
- AD. Se basa en el hecho de que todas las funciones se pueden descomponer en un número finito de operaciones para las cuales la derivada es conocida. Combinando estas derivadas se puede obtener la derivada de la función. Existen dos tipos de AD: *forward mode* y *reverse mode*.

La figura 2.2 refleja las diferencias entre estos tres métodos.

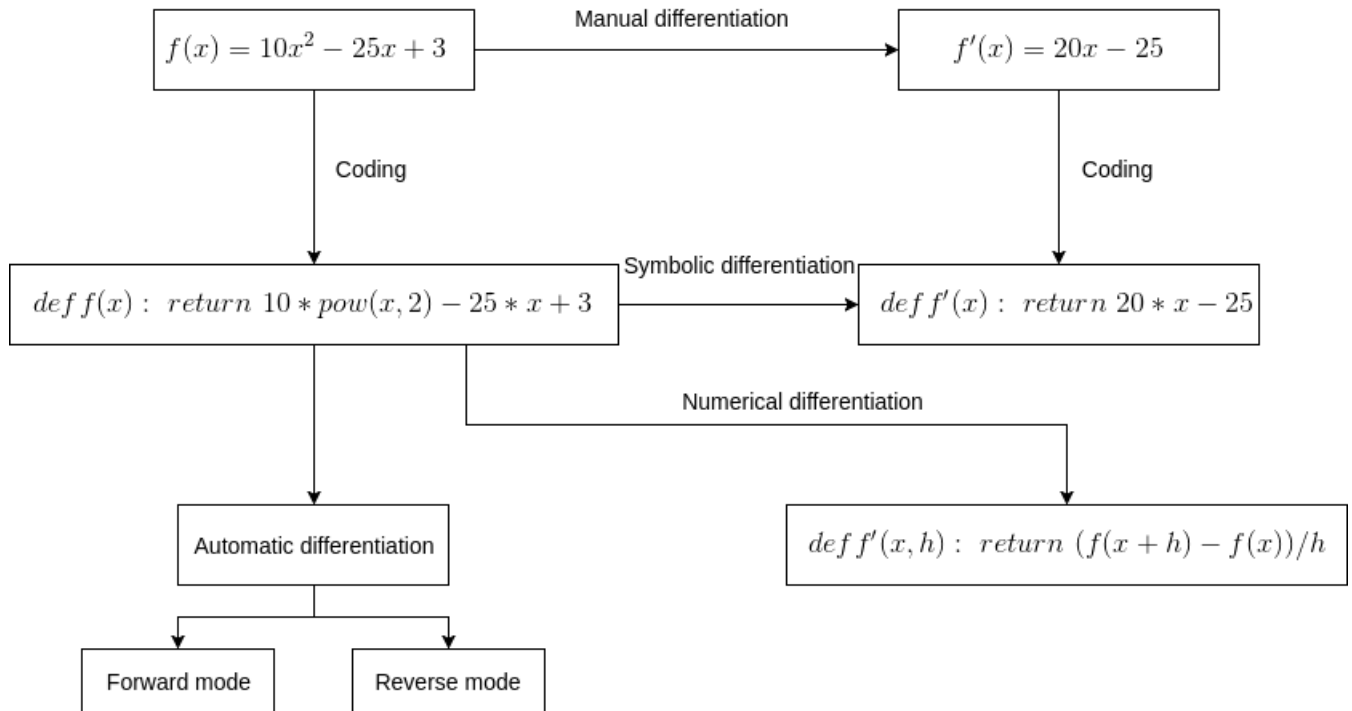


Figura 2.2: Metodologías para el cálculo de derivadas. La derivación con los diferentes modos de AD se muestra en los siguientes apartados

2.2 Automatic Differentiation forward mode

Por un lado, *forward mode* evalúa las diferentes partes de la función hacia delante y luego hace lo mismo para cada parte de la derivada hasta obtener la derivada de la función. A continuación se muestra el desarrollo para $f(x) = 10x^2 - 25x + 3$ en el punto $x = 5$:

Forward mode	
Forward evaluation trace	Forward derivative trace
$x = 5, v_0 = x$	$v'_0 = \frac{\partial v_0}{\partial x} = 1$
$v_1 = v_0^2 = 25$	$v'_1 = 2v_0 = 10$
$v_2 = 10v_1 = 250$	$v'_2 = 10v'_1 + 0v_1 = 100$
$v_3 = 25v_0 = 125$	$v'_3 = 25v'_0 + 0v_0 = 25$
$v_4 = 3$	$v'_4 = 0$
$v_5 = v_2 - v_3 = 250 - 125 = 125$	$v'_5 = v'_2 - v'_3 = 75$
$v_6 = v_5 + 3 = 128$	$v'_6 = v'_5 + 0 = 75$
$f(x) = v_6 = 128$	$f(x)' = v'_6 = 75$

2.3 Automatic Differentiation reverse mode

Por otro lado, *reverse mode* evalúa las diferentes partes de la función hacia delante pero luego añade lo que se conoce como el *reverse pass*. Este paso se basa en la regla de la cadena. Esta regla permite calcular la derivada de una función en base a las derivadas de las diferentes partes, es decir, si t depende de u , y u depende de v , la derivada de t se puede calcular mediante la fórmula:

$$\frac{\partial t}{\partial v} = \frac{\partial t}{\partial u} \frac{\partial u}{\partial v}$$

La generalización de esta fórmula quedaría como:

$$\frac{\partial t}{\partial v} = \sum_{i=1}^N \frac{\partial t}{\partial u_i} \frac{\partial u_i}{\partial v}$$

En otras palabras, para calcular la derivada de cualquier variable se requieren computar las derivadas de sus padres. El *reverse pass* empieza computando la derivada $\frac{\partial y}{\partial y} = 1$ y propaga este resultado a todas las dependencias. A continuación se muestra el desarrollo para $f(x) = 10x^2 - 25x + 3$ en el punto $x = 5$:

Reverse mode	
Forward evaluation trace	Forward adjoint trace
$x = 5, v_0 = x$	$v'_6 = \frac{\partial f(x)}{\partial v_6} = 1$
$v_1 = v_0^2 = 25$	$v'_5 = \frac{\partial f(x)}{\partial v_5} = \frac{\partial f(x)}{\partial v_6} \frac{\partial v_6}{\partial v_5} = v'_6 \cdot 1 = 1$
$v_2 = 10v_1 = 250$	$v'_4 = \frac{\partial f(x)}{\partial v_4} = 0$
$v_3 = 25v_0 = 125$	$v'_3 = \frac{\partial f(x)}{\partial v_3} = \frac{\partial f(x)}{\partial v_5} \frac{\partial v_5}{\partial v_3} = v'_5 \cdot (-1) = -1$
$v_4 = 3$	$v'_2 = \frac{\partial f(x)}{\partial v_2} = \frac{\partial f(x)}{\partial v_5} \frac{\partial v_5}{\partial v_2} = v'_5 \cdot 1 = 1$
$v_5 = v_2 - v_3 = 250 - 125 = 125$	$v'_1 = \frac{\partial f(x)}{\partial v_1} = \frac{\partial f(x)}{\partial v_2} \frac{\partial v_2}{\partial v_1} = v'_2 \cdot 10 = 10$
$v_6 = v_5 + 3 = 128$	$v'_0 = \frac{\partial f(x)}{\partial v_0} = \frac{\partial f(x)}{\partial v_1} \frac{\partial v_1}{\partial v_0} + \frac{\partial f(x)}{\partial v_3} \frac{\partial v_3}{\partial v_0} = v'_1 \cdot 2v_0 + v'_3 \cdot 25 = 75$
$f(x) = v_6 = 128$	$f(x)' = v'_0 = 75$

Así es como funciona el método *backpropagation* en las redes neuronales, las cuales necesitan las derivadas parciales para actualizar los pesos de cada una de las capas. Este método les permite reutilizar cálculos ya computados y calcular todas las derivadas de forma muy eficiente. Una de las principales ventajas del *reverse mode* frente al *forward mode* es que *reverse mode* calcula el gradiente completo (vector de n componentes), mientras que *forward mode*, para el cálculo del gradiente completo, requeriría de n evaluaciones. En este proyecto se utiliza AD *reverse mode*.

2.4 Herramientas de derivación

Como ya se ha dicho nos centraremos en la optimización de parámetros mediante AD. No obstante a continuación se listan un conjunto de paquetes *software* que permiten el cálculo de gradientes:

- *Tensorflow*¹: Librería creada y mantenida por Google. Utiliza AD *reverse mode* y actualmente es la más utilizada en temas como el *deep learning*.
- *Autograd*²: Librería escrita por Dougal Maclaurin, David Duvenaud y Matt Johnson tutorizados por Ryan P. Adams del *Harvard Intelligent Probabilistic Systems Group* (HIPS). Utiliza AD *reverse mode*.
- *Theano*³: Se trata de una librería de código abierto que originalmente fue desarrollada por un departamento de *machine learning* de la universidad de Montreal. Utiliza AD *reverse mode*.
- *Mathematica*⁴: Se trata de un programa desarrollado para las áreas científicas concebido por Stephen Wolfram y mantenido actualmente por la empresa *Wolfram Research*. Utiliza *symbolic differentiation*.

En la figura 2.3 se puede ver una comparativa de la actividad en los repositorios de *Github* de *Tensorflow*, *Theano* y *Autograd* (17/05/17).

¹ *Tensorflow*: <https://github.com/tensorflow/tensorflow>

² *Autograd*: <https://github.com/HIPS/autograd>

³ *Theano*: <https://github.com/Theano/Theano>

⁴ *Mathematica*: <https://www.wolfram.com/mathematica/>

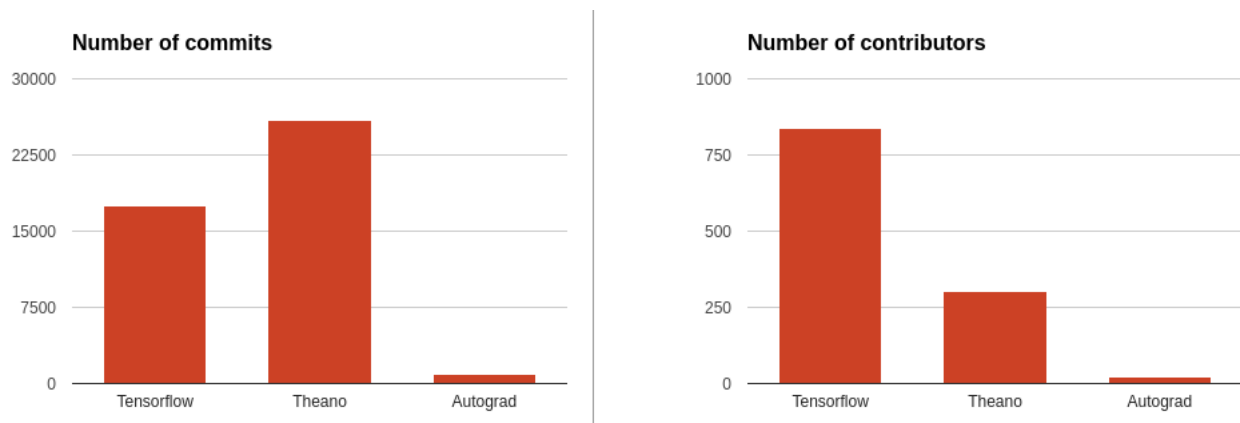


Figura 2.3: Comparativa entre las comunidades de *Tensorflow*, *Theano* y *Autograd* en *Github*

El uso de *Tensorflow* como herramienta principal para el cálculo de gradientes en este proyecto se hizo tras valorar otras como *Theano* o *Autograd*. No obstante se eligió pensando en el futuro, por su soporte a la paralelización y distribución y por la gran aceptación que ha recibido por parte de la comunidad. Actualmente (17/05/17) 836 contribuyentes y más de 17.525 *commits* en apenas su primer año de vida.

2.5 Tensorflow

Se trata de una librería de código abierto, desarrollada por *Google*, para la computación numérica mediante la utilización de grafos de flujo. Antes de la ejecución de un programa, *Tensorflow* construye un grafo de flujo donde los nodos representan operaciones matemáticas mientras que las aristas representan vectores multidimensionales de datos, a los que se les llama tensores. La construcción de este grafo le permite sacar el máximo partido tanto de las CPUs como de las GPUs del sistema donde se ejecute el programa. Así, de forma completamente transparente al programador, *Tensorflow* paraleliza todo lo que puede entre los recursos de los que disponga.

Esta librería fue diseñada en un inicio únicamente para *deep learning*, la rama del *machine learning* que estudia las redes neuronales. *Tensorflow* permite, de forma sencilla, la implementación de *Deep Neural Networks* (DNN), *Convolutional Neural Networks* (CNN) y *Recurrent Neural Networks* (RNN). No obstante, las últimas versiones de *Tensorflow* se han centrado en satisfacer al resto de la comunidad de *machine learning* intentando convertir la librería en el estándar para la programación de modelos de todas las ramas. Concretamente han desarrollado el módulo *TF Learn*⁵ que ya dispone de un conjunto de modelos listos para usar. Además, cambia la sintaxis característica de *Tensorflow* con la intención de parecerse más a la sintaxis de *Scikit-learn*⁶, una de las librerías más estables e importantes de *machine learning*.

El aspecto más interesante de esta librería para el proyecto es que implementa *AD reverse mode* de una forma elegante. El programador define un modelo indicando los parámetros como variables y prácticamente de forma automática, tras definir el algoritmo de inferencia, *Tensorflow* se encarga de calcular los gradientes y aplicarlos en la optimización.

⁵*TF Learn*: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/learn>

⁶*Scikit-learn*: <https://github.com/scikit-learn/scikit-learn>

2.6 Ejemplo de uso

En este apartado se muestra el código para la optimización de los parámetros de un modelo de regresión lineal con *Tensorflow* y con *Autograd*. Un modelo de regresión lineal viene definido por la ecuación:

$$y = wx + b$$

Donde la w es el *weight* y la b el *bias*. Mediante AD se encontrarán unos valores óptimos para estos dos parámetros que minimicen el *Mean Squared Error* (MSE):

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

En *Tensorflow*⁷ los parámetros que se quieren optimizar de una función se definen como variables (líneas 34 y 35), posteriormente se define una función de coste en base a dichos parámetros, el MSE (línea 41), y finalmente, se define el algoritmo de optimización, en este caso *Gradient Descent* (el cual se explicará más adelante) y se le indica la función a minimizar (línea 44). En las últimas líneas se define el bucle principal para el entrenamiento del modelo. Este ejemplo deriva la función de coste con el objetivo de hallar la dirección (vector gradiente) hacia un mínimo local, es decir, la dirección que reduce el MSE. Con este vector gradiente se actualizan los parámetros *weight* y *bias* del modelo (de forma transparente al programador). De esta forma, cuando se hayan hecho un número suficiente de *epochs* (pasadas sobre todos los datos) se habrán obtenido unos valores para los parámetros que minimicen la función de coste, es decir, se habrá encontrado un mínimo local.

En *Autograd*⁸ todo es más explícito que en *Tensorflow*. Se define la función de coste con los parámetros del modelo (líneas 32-35) para luego sacar gradientes en cada *epoch* e ir optimizando los parámetros *weight* y *bias* (líneas 44-47). En las figuras 2.4 y 2.5 se pueden ver los resultados para este modelo mediante las

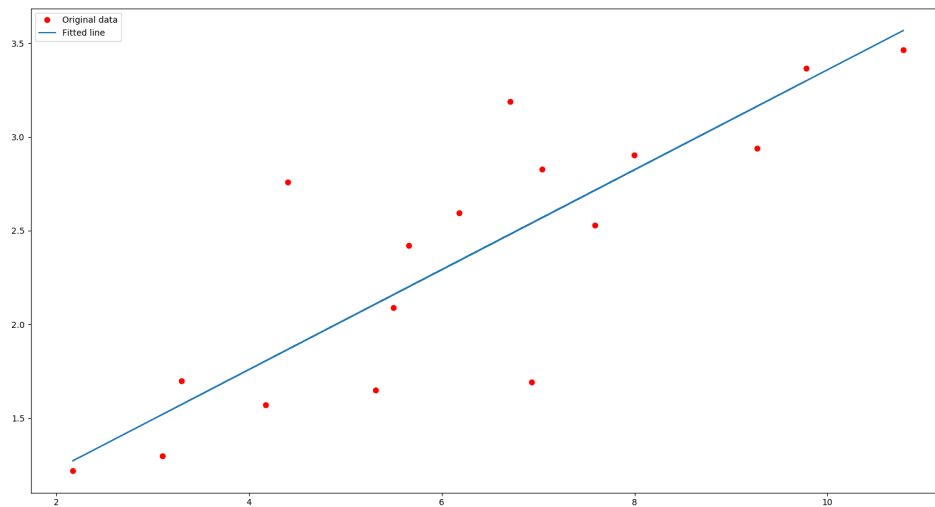


Figura 2.4: Resultado de la optimización de los parámetros de un modelo de regresión lineal con *Tensorflow* ($weight = 0.266$ y $bias = 0.695$)

herramientas *Tensorflow* y *Autograd* respectivamente. Que los resultados entre una versión y otra difieran mínimamente puede ser debido a alguna fuente de estocasticidad como la inicialización aleatoria de los parámetros del modelo.

⁷Regresión lineal en *Tensorflow*: https://github.com/bertini36/GMM/blob/master/models/linear_regression_tf.py

⁸Regresión lineal en *Autograd*: https://github.com/bertini36/GMM/blob/master/models/linear_regression_ag.py

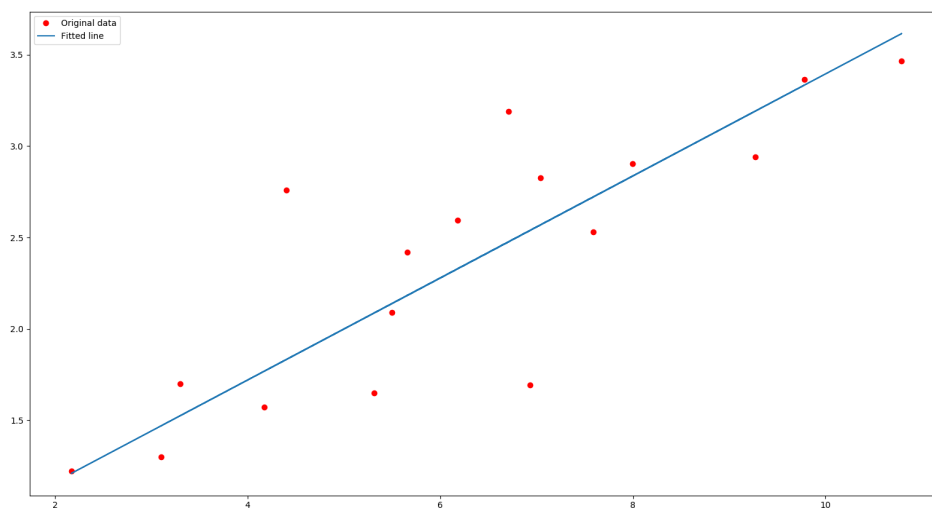


Figura 2.5: Resultado de la optimización de los parámetros de un modelo de regresión lineal con *Autograd* ($weight = 0.279$ y $bias = 0.604$)

3. Variational Inference

El cometido de este capítulo es poner al lector al día en el estado del arte de los métodos variacionales explicando los términos que se van a utilizar, implementar y discutir a lo largo de todo el proyecto. Con ello se intenta justificar cada uno de los pasos que se han dado en los últimos años en la inferencia de modelos probabilísticos.

3.1 Probabilistic machine learning

En las últimas décadas el estudio del *machine learning* ha provocado la aparición de una gran variedad de algoritmos para solventar un amplio conjunto de problemas que abarca desde la conducción de vehículos autónomos o la diagnosis médica hasta el reconocimiento del habla o la clasificación de usuarios para campañas de *marketing*. Estos algoritmos se basan principalmente en la construcción de un modelo que describa lo más fielmente posible los datos.

Un modelo es una descripción compacta de unos datos que nos permite hacer predicciones sobre futuras muestras. La principal diferencia entre un modelo de *machine learning* convencional y un modelo probabilístico es que este último nos permite **modelar la incertidumbre**, es decir, nos permite saber con qué probabilidad se cumplirá la predicción. Este aspecto puede ser muy valioso en áreas como la medicina o la economía donde el riesgo de tomar una decisión u otra puede ser perjudicial para la salud de un ser humano o suponer una pérdida económica.

Por un lado, este tipo de modelos utiliza la teoría de probabilidad para **modelar información a**

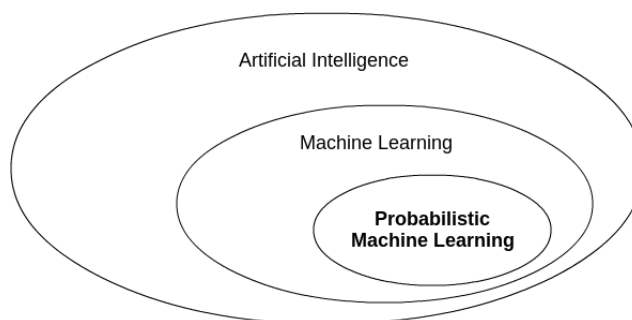


Figura 3.1: Situación del *Probabilistic Machine Learning* en el área de la inteligencia artificial

priori, de esta forma el algoritmo no se basa únicamente en los datos de la muestra. También permiten el **uso de diferentes datasets** de los que aprender, son muy útiles cuando se quieren modelar probabilidades de muchas cosas o se dispone de una **cantidad reducida de información** y pueden definir modelos complejos con la cantidad de variables aleatorias que se requieran. Estos modelos **soportan online learning**, es decir, no hace falta volver a entrenar todo el modelo cada vez que se obtengan nuevos datos sino que simplemente se actualizan las probabilidades. Por otro lado, son muy útiles para la toma de decisiones, cuando se requiere una **explicación robusta del modelo**. Además pueden ser **modelos generativos** lo cual, gracias a las distribuciones que se modelan, permite generar nuevos datos simulando valores de cualquier variable aleatoria del modelo. A diferencia de los modelos discriminantes que solo modelan la probabilidad de la variable a predecir, un modelo generativo es un modelo probabilístico completo sobre todas las variables (observadas y latentes), lo que permite dar respuesta a múltiples preguntas.

3.2 Bayesian Inference

Bayesian Inference trata de desvelar la estructura oculta en los datos que no se puede observar directamente. En los métodos tradicionales de *machine learning* los parámetros del modelo son valores que son determinados por algoritmos de optimización mediante la minimización o maximización de una función de error. El punto de vista bayesiano es algo diferente, para un bayesiano todos los parámetros desconocidos son descritos mediante distribuciones de probabilidad y la observación de evidencias permite la actualización de estas distribuciones utilizando la regla de Bayes.

3.2.1 Regla de Bayes

Lo primero que hay que tener claro para entender en qué consiste *Bayesian Inference* es la regla de Bayes. La regla de Bayes nos indica como debe actualizarse una probabilidad sobre un suceso después de observar evidencias sobre éste. Desde un punto de vista bayesiano, no hay diferencias entre parámetros y variables observadas, ambas son variables aleatorias. Se utilizará x para denominar a las variables observadas y θ para las variables latentes.

El resumen de esta fórmula sería el siguiente: inicialmente se tiene una creencia (*prior*) sobre un evento θ ($p(\theta)$), por ejemplo, que la altura de la población de Barcelona sigue una distribución *Normal*. Después se observa la evidencia (x), una muestra de las alturas de la población de Barcelona. En función de esta evidencia la creencia sobre θ debe cambiar, es decir, la distribución *Normal* que describía la altura de la población de Barcelona se actualizará. Este cambio lo refleja el *posterior* ($p(\theta|x)$).

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

A continuación se explica qué son cada uno de los términos de la fórmula siendo x y θ los datos y los parámetros del modelo respectivamente:

- **Posterior** $p(\theta|x)$: Es la probabilidad de los parámetros dados los datos. En otras palabras, la probabilidad de que el modelo con los parámetros θ haya generado los datos x . El *posterior* es la incógnita en *Bayesian Inference*.
- **Likelihood** $p(x|\theta)$: Es la probabilidad de los datos dados los parámetros asumiendo que éstos son modelados por una distribución parametrizada. La forma de calcularla depende del modelo.
- **Prior** $p(\theta)$: Es la probabilidad de los parámetros. En este factor de la fórmula es donde se refleja el conocimiento a priori que se tiene sobre el modelo, la información que se tiene, antes de observar ningún dato, sobre el comportamiento de las variables latentes.
- **Evidence** $p(x)$: Se trata de la evidencia de los datos. Se calcula como $\int p(x, \theta) d\theta$. Normalmente no se puede calcular y eso supone el principal problema de *Bayesian Inference*.

El producto $p(x|\theta)p(\theta)$ también se conoce como la probabilidad conjunta (*joint probability*): $p(\theta, x)$ y se trata de la probabilidad que se representa mediante los *probabilistic graphical models*.

Cuando se comparan modelos probabilísticos, si se utiliza la *likelihood* para estimar los parámetros θ se conoce como *Maximum Likelihood Estimation* (MLE), mientras que si además se tiene en cuenta el *prior* se conoce como *Maximum A Posteriori estimation* (MAP). No obstante estos métodos sólo permiten estimar la media o mediana del *posterior* y tal vez el objetivo requiera modelar la incertidumbre o generar nuevos datos, lo que implicaría tener que conocer la distribución del *posterior*. Como se presentará más adelante métodos como *Variational Inference* (VI) o *Markov Chain Monte Carlo* (MCMC) permiten modelar esta distribución. A diferencia de MLE y MAP, el hecho de tener en cuenta la evidencia del modelo ($p(x)$), es lo que permite a estas estrategias calcular la distribución *posterior*.

Online learning

Tal como se ha dicho, el *online learning* es una aptitud importante de los modelos probabilísticos. Viendo la definición de la regla de Bayes se puede apreciar el soporte al *online learning* que permiten este tipo de modelos. Al final es un proceso iterativo de actualización de unas creencias (*prior*) en base a unas evidencias (x) donde el *posterior* de una iteración será el *prior* de la siguiente.

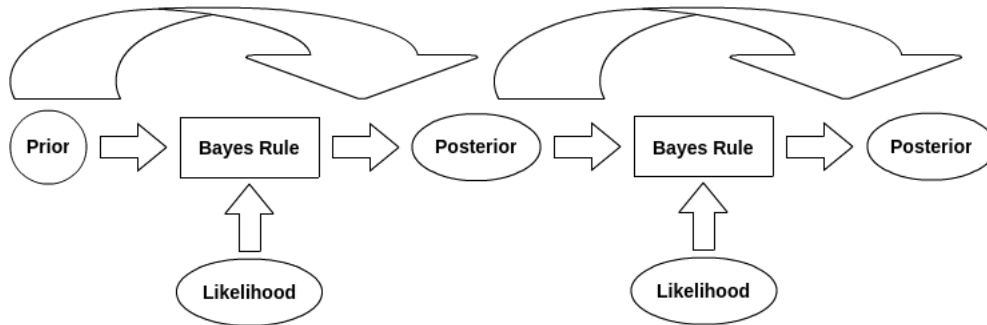


Figura 3.2: *Online learning* con la regla de Bayes

3.2.2 Aproximación del posterior

Los algoritmos de inferencia del *posterior* nos permiten analizar información bajo ciertas suposiciones (*priors*) infiriendo la estructura oculta que mejor describe nuestros datos. Cuando todas las relaciones entre las variables aleatorias del modelo son conjugadas¹, es decir, cuando la distribución conjunta del modelo es de la misma forma que el *prior*, el *posterior* se puede calcular de forma analítica. Este tipo de inferencia se conoce como *exact inference*. En el caso contrario, el problema de esta fórmula reside en el cálculo de la evidencia. Para muchos modelos de interés como *Gaussian Mixture Model* (GMM) o *Latent Dirichlet Allocation* [35], el cálculo del *posterior* es intratable computacionalmente por culpa de la integral sobre todas las variables latentes de los datos que conlleva el cálculo de la evidencia. Para estos modelos se requiere de otra estrategia para modelar el *posterior*.

En *probabilistic machine learning* se utilizan variables latentes para describir la estructura oculta en los datos, se modelan relaciones entre las variables observadas y latentes mediante distribuciones de probabilidad y se usan algoritmos de inferencia para estimar el *posterior*, es decir, la distribución condicional de las variables latentes dadas las variables observadas. Debido a que en la mayoría de ocasiones se trabaja sobre espacios de muchas dimensiones, con una cantidad muy elevada de datos o con modelos no conjugados, el cálculo del *posterior* ($p(\theta|x)$), es impracticable analítica y computacionalmente, por ello se han desarrollado métodos de inferencia para aproximar esta distribución.

El concepto de *Bayesian Inference* viene del conjunto de técnicas que se han desarrollado para la aproximación del *posterior* y es uno de los problemas centrales actuales en estadística bayesiana. Existen dos ramas algorítmicas para la aproximación del *posterior* en *Bayesian Inference*:

1. MCMC: *Sampling approximate inference*.
2. VI: *Structural approximate inference*.

Por un lado, MCMC se basa en la construcción de una cadena de *Markov* sobre las variables latentes siendo su distribución estacionaria el *posterior*. Después se ejecuta la cadena hasta que llega al punto de equilibrio. Finalmente el resultado de las muestras de la cadena de *Markov* en su tramo estacionario, són las muestras del *posterior*. Los algoritmos más conocidos de esta familia son *Metropolis-Hastings*, *Gibbs Sampling* y *Hamiltonian Monte Carlo* (HMC).

Por otro lado, VI aproxima el *posterior* creando una aproximación analítica, el modelo variacional, el cual va ajustando con el objetivo de reducir la diferencia con el *posterior*. En esta familia de algoritmos el problema deja de ser de aproximación para ser de optimización.

¹Relación de conjugación: https://en.wikipedia.org/wiki/Conjugate_prior

Este proyecto se centra en el estudio del comportamiento de las diferentes estrategias existentes en la rama algorítmica de VI.

3.2.3 Probabilistic graphical models

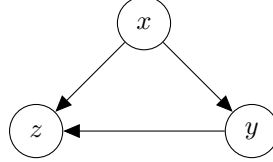
En el ámbito bayesiano un modelo representa una probabilidad conjunta sobre todas las variables aleatorias del problema.

$$p(x_1, \dots, x_n, \theta_1, \dots, \theta_m)$$

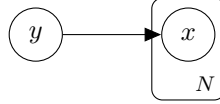
Los *probabilistic graphical models* nos aportan un lenguaje visual para representar estas probabilidades conjuntas, es decir, permiten representar las suposiciones sobre nuestros datos y su estructura oculta. En este proyecto utilizaremos *bayesian networks* para representar los modelos. Esta representación en forma de grafo dirigido proporciona una forma de representar las dependencias entre variables y permite determinar la complejidad de los algoritmos puesto que algunos de éstos se pueden ver como acciones sobre el grafo [36]. En este proyecto nos centraremos en modelos probabilísticos donde se expresará una distribución de probabilidad como un *Directed Acyclic Graph* (DAG).

Por ejemplo, el *probabilistic graphical model* de la distribución conjunta:

$$p(x, y, z) = p(z|x, y)p(y|x)p(x)$$



En este contexto $p(z|x, y)$ representa la probabilidad condicional de z dado el valor de x e y , y $p(y|x)$ representa la probabilidad condicional de y dado el valor de x . En este tipo de diagramas también existen unos componentes llamados *plates*, como por ejemplo en el siguiente *probabilistic graphical model*:

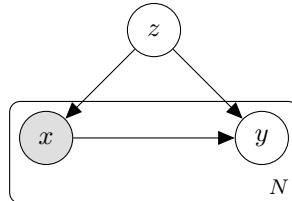


Esta notación indica un vector de n variables aleatorias x y sería la representación gráfica de la siguiente probabilidad conjunta:

$$p(x, y) = p(y) \prod_{i=1}^N p(x_i|y)$$

Variables locales y globales

En un modelo probabilístico se pueden distinguir dos tipos de variables aleatorias: globales y locales. Una variable global es aquella que comparten todos los ejemplos del *dataset* mientras que una variable local es propia de cada ejemplo. Por ejemplo, en el siguiente *probabilistic graphical model* la variable y es una variable local mientras que la variable z es una variable global. Cuando un nodo aparece oscurecido quiere decir que se trata de una variable observada.



3.3 Variational Inference

En este proyecto nos hemos centrado en las diferentes estrategias variacionales para la aproximación del *posterior*. En los siguientes apartados se explica la evolución que ha experimentado VI para solventar los problemas que han ido surgiendo.

3.3.1 Estrategia

Como ya se ha comentado VI consiste en definir una distribución, $q(\theta|\lambda)$, cuyos parámetros λ se irán optimizando con el objetivo de reducir sus diferencias con el *posterior* $p(\theta|x)$. Esta nueva distribución es conocida como modelo variacional y el *posterior* como modelo probabilístico. En resumen, el objetivo es optimizar los parámetros λ del modelo variacional $q(\theta|\lambda)$ de forma que se vayan reduciendo las diferencias con el modelo probabilístico $p(\theta|x)$. A λ también se lo conoce por el nombre de parámetros variacionales. En la figura 3.3 se puede ver un esquema de la estrategia que sigue VI.

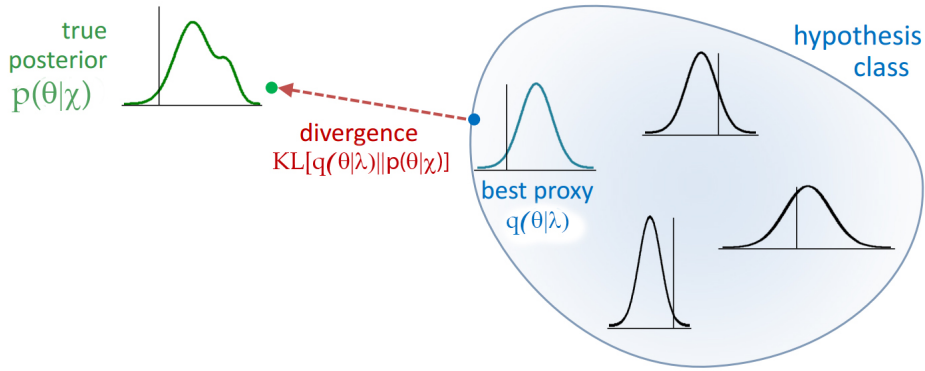


Figura 3.3: Estrategia de VI (Fuente: [37])

Kullback-Leibler divergence

Calcular la distancia Euclídea entre los parámetros de las distribuciones para establecer la similitud entre ambas no es una medida fiable puesto que se están comparando distribuciones y no puntos. Imaginemos una distribución *Normal* con media 0 y varianza 5, $\mathcal{N}(0, 5)$, y otra con media 5 y varianza 5, $\mathcal{N}(5, 5)$. Estas dos distribuciones son muy parecidas pero se encuentran a una distancia Euclídea de 5. Si ahora comparamos la primera *Normal*, $\mathcal{N}(0, 5)$, con otra con media 2 y varianza 7, $\mathcal{N}(2, 7)$, parece que son bastante diferentes pero la distancia Euclídea entre ellas es 4. Por esta razón debemos utilizar otra medida de similitud diferente a la distancia Euclídea: la *Kullback-Leibler divergence* (KL) [38].

La KL es una divergencia (una distancia no simétrica), es decir, no es lo mismo calcular la $KL[p(\theta|x)||q(\theta|\lambda)]$ (*forwards* KL) que la $KL[q(\theta|\lambda)||p(\theta|x)]$ (*reverse* KL). El hecho de usar una u otra da lugar a algoritmos diferentes: *Expectation Propagation* usa *forwards* KL mientras que VI utiliza *reverse* KL. En general *Expectation Propagation* es más costoso computacionalmente. KL cuantifica la información que se pierde al aproximar una distribución con otra. Se basa en el concepto de entropía. La entropía mide la cantidad de información que poseen los datos y se define de la siguiente manera:

$$H = - \sum_{i=1}^N p(x_i) \ln p(x_i)$$

La definición de la KL se obtiene en base a la diferencia de entropía entre las 2 distribuciones y queda de la siguiente manera:

$$KL[q(\theta|\lambda)||p(\theta|x)] = \int q(\theta|\lambda) (\ln q(\theta|\lambda) - \ln p(\theta|x)) = \int q(\theta|\lambda) \ln \frac{q(\theta|\lambda)}{p(\theta|x)}$$

Esta divergencia permite encontrar la similitud real entre dos distribuciones de probabilidad y es la medida de similitud que minimiza el algoritmo de VI. En las figuras 3.4 y 3.5 [39] se pueden ver las diferencias entre la aproximación de *forwards* y *reverse* KL de una distribución bimodal (distribución con dos modos) y una distribución unimodal (distribución con un único modo) respectivamente. La parte azul representa la distribución a aproximar y la roja la aproximación.

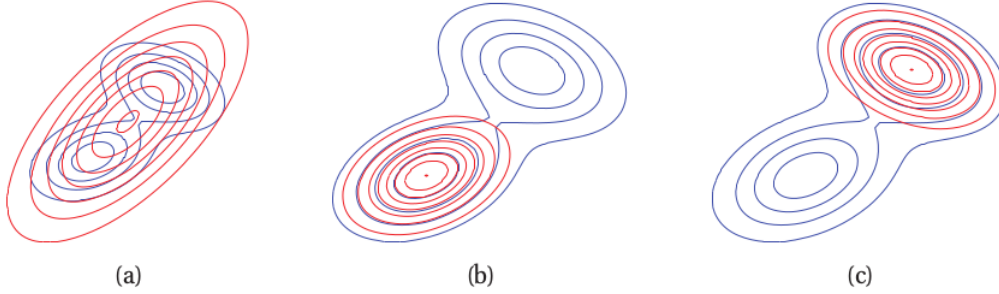


Figura 3.4: La figura a es una aproximación de *forwards* KL y las figuras b y c son aproximaciones de *reverse* KL

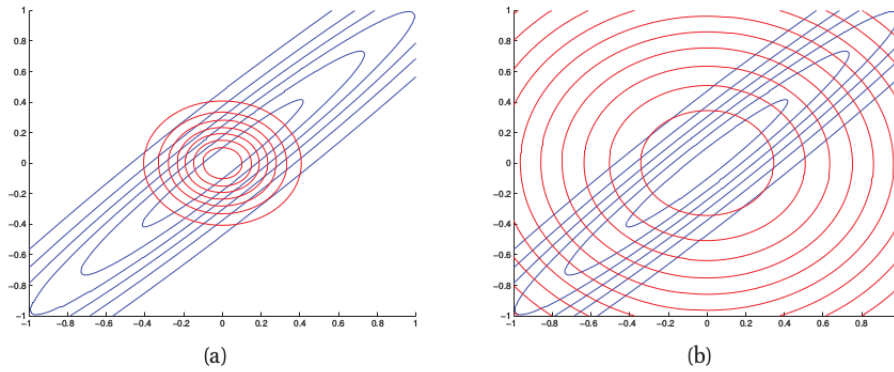


Figura 3.5: La figura a es una aproximación de *forwards* KL y la figura b es una aproximación de *reverse* KL

Mean-field assumption

Con el objetivo de definir una distribución tratable sobre todas las variables latentes para aproximar el *posterior* se simplifica la optimización del modelo variacional suponiendo que se trata de un modelo factorizado. Esto implica suponer que $q(\theta|\lambda)$ esta compuesta por un conjunto de distribuciones $q_i(\theta_i|\lambda_i)$ (de la *Exponential Family*²). Cada una de estas distribuciones tiene sus parámetros λ_i los cuales son optimizables individualmente.

$$q(\theta|\lambda) = \prod_i q_i(\theta_i|\lambda_i)$$

²*Exponential Family*: https://en.wikipedia.org/wiki/Exponential_family

3.3.2 Procedimiento

En las siguientes líneas se demuestra que para aproximar el *posterior*, que como ya se ha comentado es intratable computacionalmente, se requiere minimizar la KL entre el modelo variacional $q(\theta|\lambda)$ y el modelo probabilístico $p(\theta|x)$. Partiendo de la regla de Bayes tenemos:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \Rightarrow p(x) = \frac{p(x, \theta)}{p(\theta|x)} \Rightarrow$$

$$\ln p(x) = \int q(\theta|\lambda) \ln \frac{q(\theta|\lambda)}{p(\theta|x)} d\theta + \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta$$

Se puede ver la derivación completa en el apéndice A.1.1 y cada uno de los factores representa:

- $\int q(\theta|\lambda) \ln \frac{q(\theta|\lambda)}{p(\theta|x)} d\theta$: $KL[q(\theta|\lambda)||p(\theta|x)]$.
- $\int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta$: *Model Evidence Lower Bound* ($ELBO(q(\theta|\lambda), p(x, \theta))$).

De forma que queda de la siguiente manera:

$$\ln p(x) = KL[q(\theta|\lambda)||p(\theta|x)] + ELBO(q(\theta|\lambda), p(x, \theta))$$

Con esta demostración se llega a que minimizar $KL[q(\theta|\lambda)||p(\theta|x)]$ es equivalente a maximizar la $ELBO(q(\theta|\lambda), p(x, \theta))$. El procedimiento de VI se centra en maximizar la función *Evidence Lower Bound* (ELBO) dado que es más asequible computacionalmente puesto que no depende del *posterior*. Al final lo que se ha hecho es convertir una integral intratable en la *expectation* de una distribución conocida. En la figura 3.6 se muestra un esquema que resume la estrategia de dicha inferencia. La ELBO es una medida que estima la

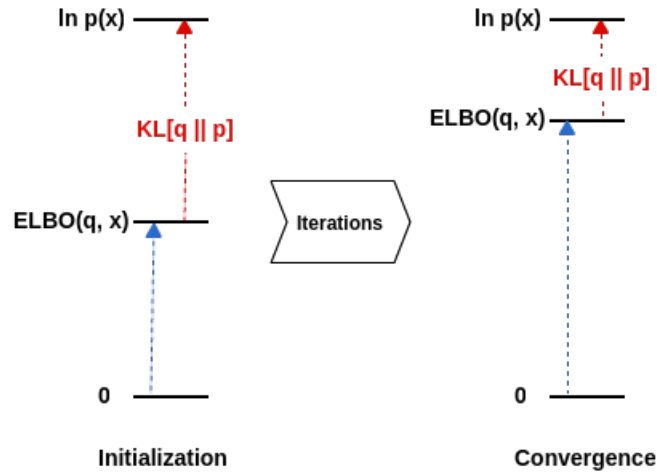


Figura 3.6: Maximizar la ELBO es equivalente a minimizar la distancia entre el modelo variacional y el probabilístico

calidad del modelo, cuanto más elevado sea su valor más acertado es el modelo inferido. VI utiliza la ELBO como condición de parada del algoritmo. Cuando se llega a un conjunto de iteraciones donde el valor de la ELBO no aumenta más quiere decir que se han encontrado unos parámetros λ para el modelo variacional que consiguen aproximar fielmente el modelo probabilístico (el *posterior*). En el caso de que $q(\theta|\lambda)$ convergiera con $p(\theta|x)$, la $KL[q(\theta|\lambda)||p(\theta|x)]$ sería 0 y la $ELBO(q(\theta|\lambda), p(x, \theta))$ coincidiría con la evidencia ($p(x)$). De aquí que la ELBO se utilice para medir la calidad del modelo o como condición de parada del algoritmo.

También podemos reescribir la ELBO como (véase el apéndice A.1.2):

$$ELBO(q(\theta|\lambda), p(x, \theta)) = \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$$

Donde cada termino es:

- $\mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta))$: *Expectation* de la probabilidad conjunta del modelo.
- $\mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$: Entropía de *Shannon*.

Y si ahora se tiene en cuenta la *mean-field assumption* comentada anteriormente, la ELBO queda de la siguiente manera:

$$\begin{aligned}
ELBO(q(\theta|\lambda), p(x, \theta)) &= \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta \\
&= \int \prod_i q_i(\theta_i|\lambda_i) \left(\ln p(x, \theta) - \sum_i \ln q_i(\theta_i|\lambda_i) \right) d\theta \\
&= \mathbb{E}_{q(\theta|\lambda)} \ln p(x, \theta) - \prod_i \mathbb{E}_{q(\theta_i|\lambda_i)} \ln q(\theta_i, \lambda_i)
\end{aligned}$$

3.3.3 Algoritmo

El algoritmo básico de VI, concretamente conocido como *Mean-Field Variational Inference* (MFVI), quedaría de la siguiente manera:

Algorithm 1: MFVI algorithm.

```

1 Initialize  $\lambda$  randomly
2 repeat
3   | Update  $\lambda_i$  of every  $q_i(\theta_i|\lambda_i)$ 
4   | Recalculate ELBO
5 until ELBO grows no more;
```

Este algoritmo representa la idea básica de VI. En la práctica hay que tener en cuenta más cosas para su correcto funcionamiento. Para empezar, un modelo variacional puede componerse de variables locales y variables globales y la actualización de éstas debe hacerse de una forma concreta. También se debe definir el método de optimización: *coordinate ascent*, *gradient ascent*, *stochastic gradient ascent*, ... El método de optimización elegido no influye en la estructura básica del algoritmo de VI, únicamente cambia la forma de obtener el nuevo valor de los parámetros λ del modelo variacional en cada una de las iteraciones del algoritmo. En todo esto se entrará en más profundidad en los próximos apartados.

3.4 Coordinate Ascent Variational Inference

A continuación se explica el método más tradicional para la inferencia de modelos probabilísticos: *Coordinate Ascent Variational Inference* (CAVI). Para la implementación de este tipo de inferencia se requiere de conocimientos de probabilidad y cálculo para la derivación de las actualizaciones de cada parámetro λ del modelo variacional y para la derivación de la ELBO. Como ya se ha mencionado anteriormente, cuando el modelo es completamente conjugado, el *posterior* se puede calcular analíticamente, es decir, sin necesidad de aproximarlos (*exact inference*). No obstante, si la cantidad de datos disponibles para este modelo es muy grande, la implementación de este cálculo analítico del *posterior* se vuelve impracticable computacionalmente debido a la operabilidad con matrices muy grandes en memoria. Por tanto, en este caso y en el caso de modelos no conjugados es acertado aproximar el *posterior* mediante VI.

La derivación de las actualizaciones analíticas de los parámetros del modelo variacional se puede realizar de dos formas: por derivación genérica o mediante las propiedades de la *Exponential Family*.

3.4.1 Derivación genérica

Este tipo de derivación se basa en la siguiente fórmula (suponiendo la *mean-field assumption*):

$$q(\theta_i|\lambda_i) \propto \exp(\mathbb{E}_{q(\theta_{\neq i}|\lambda_{\neq i})}(\ln(p(x, \theta))))$$

El origen de esta fórmula se puede ver en el apéndice A.1.4. Esta derivación se haría para cada variable del modelo variacional θ_i y tras la derivación se podría deducir qué distribución debe tener dicho parámetro en el modelo variacional y cómo actualizarlo.

Ejemplo de la actualización de un parámetro variacional mediante derivación genérica

Supongamos el siguiente factor $q(\pi|\lambda_\pi)$, el cual forma parte de un modelo variacional donde también están las variables c y μ (una mixtura de Gaussianas con varianza fija). El resultado de la derivación sería:

$$q(\pi|\lambda_\pi) \propto \exp\left(\sum_{k=1}^K (\alpha_k - 1) + \sum_{n=1}^N \mathbb{E}_{q(c_n|\lambda_{c_n})} \mathbb{I}(c_n = k) \ln(\pi_k)\right)$$

Se puede ver la derivación completa en el apéndice A.1.3.

$$q(\pi|\lambda_\pi) = \text{Dir}\left(\alpha_k + \sum_{n=1}^N \mathbb{E}_{q(c_n|\lambda_{c_n})} \mathbb{I}(c_n = k)\right)$$

Durante la derivación se van eliminando los términos de la ecuación que no dependen del parámetro sobre el que se está derivando. Con esta derivación se concluye que la variable π del modelo variacional es una distribución *Dirichlet* y que su parámetro se actualiza en cada iteración de la siguiente manera:

$$\lambda_\pi = \alpha_k + \sum_{n=1}^N \mathbb{I}(\lambda_{c_n} = k)$$

La asociación que hace el estadístico para reconocer que esta distribución tendrá la forma de una distribución *Dirichlet* es en base a los *sufficient statistics* (se explicará en el siguiente apartado). En resumen, el estadístico es capaz de reconocer a la distribución por el término $\ln(\pi_k)$, el cual es el *sufficient statistic* de una distribución *Dirichlet* (se puede ver en el último paso de la derivación presentada en el apéndice A.1.3).

3.4.2 Derivación mediante las propiedades de la Exponential Family

Otra forma de obtener las actualizaciones de los parámetros variacionales es derivarlos mediante las propiedades de la *Exponential Family*. A esta familia pertenecen todas las distribuciones que se pueden escribir de la forma:

$$p(x|\theta) = h(x)\exp(\eta(\theta)t(x) - a(\eta(\theta)))$$

- $h(x)$: *Base measure*.
- $\eta(\theta)$: *Natural parameters* (solo depende de los parámetros).
- $t(x)$: *Sufficient statistics* (solo depende de los datos). Permite saber la forma que tiene la distribución. Describe el espacio posible para los parámetros de la distribución.
- $a(\eta(\theta))$: Cumulante. Es un normalizador.

Esta familia permite establecer relaciones de conjugación entre distribuciones. Cuando al crear la distribución conjunta en base a otras dos distribuciones, los *natural parameters* de una permiten alguna simplificación en la formulación junto a los *sufficient statistics* de la otra, se dice que la primera distribución es conjugada de la segunda. De esta manera, la distribución resultante tendrá la misma forma que la primera distribución. Los modelos conjugados, como ya se ha dicho, gracias a estas simplificaciones, permiten calcular el posterior de forma analítica (*exact inference*).

Distribución Dirichlet en Exponential Family

A modo de ejemplo se va a mostrar la formulación de una distribución *Dirichlet* en forma exponencial. La distribución *Dirichlet* modela las proporciones de pertenencia a un conjunto de K clases, es la generalización de la distribución *Beta*³ y es el *conjugate prior* de la distribución *Categorical*. Recibe como parámetro un vector α de k valores. Cuanto mayor sea el valor de un α_k concreto más seguro se está de la probabilidad de k y cuanto más grande sea ese valor con respecto al resto de α_{k_s} más probabilidad se le da a esa clase. Como salida, esta distribución da un vector de proporciones a cada una de las clases.

³*Beta distribution*: https://en.wikipedia.org/wiki/Beta_distribution

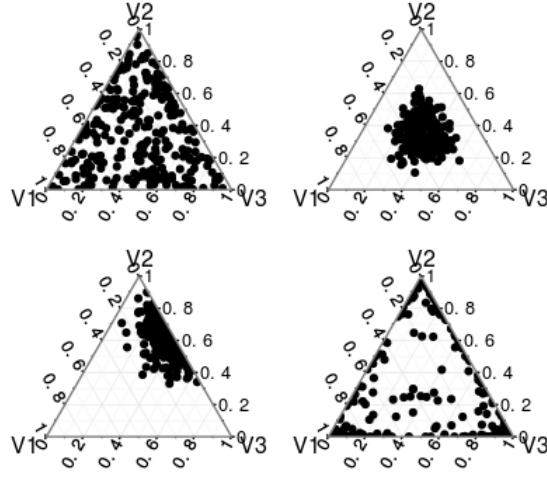


Figura 3.7: Ejemplo de una distribución *Dirichlet* de 3 clases. (Fuente: *stackexchange*)

En la figura 3.7 se puede ver el resultado del muestreo de una distribución *Dirichlet* de 3 clases para diversas configuraciones del vector α . La primera configuración se correspondería a $\alpha_1 = 1$, $\alpha_2 = 1$ y $\alpha_3 = 1$ y esto provoca que los puntos generados estén uniformemente distribuidos por el espacio. En la segunda configuración se tiene $\alpha_1 = 10$, $\alpha_2 = 10$ y $\alpha_3 = 10$, es decir, ya se está más seguro en el resultado y por tanto los puntos se distribuyen en la zona central del triángulo dando lugar a probabilidades similares para las 3 clases. La tercera configuración se correspondería a $\alpha_1 = 1$, $\alpha_2 = 10$ y $\alpha_3 = 5$, como se ve en la imagen los puntos generados están entre $V2$ y $V3$ pero más cerca de $V2$ dado que su α es mayor. Finalmente, la última configuración se corresponde con los valores $\alpha_1 = 0.2$, $\alpha_2 = 0.2$ y $\alpha_3 = 0.2$. Cuando el valor de las α_s es menor que 0 se puede considerar un anti-peso que provoca que las muestras se generen en los bordes del triángulo.

Probability Density Function (PDF):

$$p(\pi|\alpha) = \frac{\tau(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \tau(\alpha_i)} \prod_{k=1}^K \pi_k^{\alpha_k - 1}$$

Formulación en *Exponential Family*:

$$p(\pi|\alpha) = \exp\left(\ln\left(\tau\left(\sum_{k=1}^K \alpha_k\right)\right) - \ln\left(\prod_{k=1}^K \tau(\alpha_i)\right) + \begin{bmatrix} \alpha_1 - 1 \\ \dots \\ \alpha_k - 1 \end{bmatrix}^T \begin{bmatrix} \ln(\pi_1) \\ \dots \\ \ln(\pi_k) \end{bmatrix}\right)$$

Se puede ver la derivación completa en el apéndice A.1.5.

$$\text{Exponential Family: } p(x|\theta) = h(x)\exp(\eta(\theta)t(x) - a(\eta(\theta)))$$

- $h(\pi) = 1$
- $a(\eta(\alpha)) = \ln\left(\tau\left(\sum_{k=1}^K \alpha_k\right)\right) - \ln\left(\prod_{k=1}^K \tau(\alpha_i)\right)$
- $\eta(\alpha) = \begin{bmatrix} \alpha_1 - 1 \\ \dots \\ \alpha_k - 1 \end{bmatrix}$
- $t(\pi) = \begin{bmatrix} \ln(\pi_1) \\ \dots \\ \ln(\pi_k) \end{bmatrix}$

Que el *sufficient statistic* de una distribución *Dirichlet* sea $\ln(\pi_i)$ no es casualidad, está indicando que los parámetros de la distribución tienen que ser números reales positivos.

En este proyecto se utiliza este tipo de derivación para obtener las fórmulas analíticas para la actualización de los parámetros variacionales. En el siguiente capítulo se desarrollarán estas derivaciones para los parámetros del modelo GMM.

Actualización de un parámetro variacional mediante las propiedades de la Exponential Family

En primer lugar es necesario definir la distribución *complete conditional* de una variable aleatoria. La distribución *complete conditional* de una variable latente es la probabilidad de esa variable y las variables directamente relacionadas con ella dadas el resto de variables latentes y los datos.

$$p(\theta_i, \text{sons}(\theta_i) | \theta_{\neq i}, x)$$

Las actualizaciones de cada parámetro variacional se obtienen de aislarlo de la igualdad formada por los *natural parameters* de la distribución variacional de ese parámetro y la *expectation* de los *natural parameters* de la distribución *complete conditional* de ese parámetro.

$$\eta_{q(i|\lambda_i)} = \mathbb{E}_{q(\text{sons}(i) | \lambda_{\text{sons}(i)})} \eta_{p(i|\text{sons}(i))}$$

En el siguiente capítulo se muestran las derivaciones, siguiendo esta metodología, para la obtención de las actualizaciones analíticas de los parámetros variacionales del modelo GMM.

3.4.3 Algoritmo

El algoritmo CAVI actualiza, en cada iteración, cada uno de los parámetros variacionales utilizando las fórmulas analíticas derivadas. Su algoritmo quedaría de la siguiente manera:

Algorithm 2: CAVI algorithm.

```

1 Initialize  $\lambda$  randomly
2 repeat
3   | Update each variational parameter  $\lambda$  with its derived formula
4   | Recalculate ELBO
5 until ELBO grows no more;
```

Hay que tener en cuenta que si el modelo tiene variables locales y globales, como es el caso de GMM, se tendrían que distinguir sus actualizaciones. En la figura 3.8 se puede apreciar el efecto de escalera que provoca la actualización individual de cada parámetro realizado por el método de inferencia *coordinate descent/ascent*.

3.4.4 Ejemplo

A continuación se muestra un ejemplo, mediante una función simple en un espacio de 2 dimensiones, de como se desarrollaría una optimización mediante *coordinate descent*. La función a optimizar es:

$$f(x, y) = 5x^2 - 6xy + 5y^2$$

Tan solo se muestra el proceso de dos iteraciones del algoritmo. Es decir, una optimización de la variable x y una optimización de la variable y .

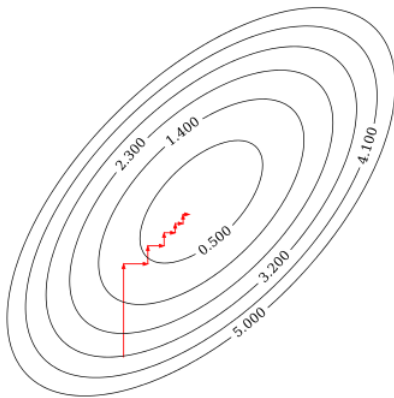


Figura 3.8: Representación gráfica de la búsqueda de un mínimo relativo realizada con *coordinate descent* en un espacio de 2 dimensiones

- Se fija el valor de x a -0.5 (aleatorio)

$$f(y) = 5y^2 + 3y + 1.25$$

- Se encuentra el mínimo de y

$$\partial f(y) = 10y + 3$$

$$y = -0.3$$

- Se fija y con el mínimo encontrado: -0.3

$$f(x) = 5x^2 + 1.8x + 0.45$$

- Se encuentra el mínimo de x

$$\partial f(x) = 10x + 1.8$$

$$x = -0.18$$

3.5 Gradient Ascent Variational Inference

Otra alternativa para la inferencia de los parámetros variacionales es *Gradient Ascent Variational Inference* (GAVI). La diferencia reside en el hecho de que la actualización de los parámetros variacionales no se hace con unas fórmulas analíticamente cerradas sino que es un proceso mucho más exploratorio. GAVI se basa en el algoritmo *gradient descent/ascent*.

3.5.1 Gradient ascent

Gradient Ascent tiene como objetivo maximizar una función de coste $\mathbb{C}(\lambda)$ parametrizada por los parámetros del modelo, λ . El algoritmo va optimizando estos parámetros λ en la dirección del gradiente (en el caso de *gradient descent*, en la dirección opuesta del gradiente) de la función objetivo $\nabla_{\lambda}\mathbb{C}(\lambda)$.

En nuestro caso nos interesa *gradient ascent* puesto que lo que se busca con VI es maximizar una función: la ELBO. El *learning rate* $\rho > 0$ determina el tamaño del paso en la dirección del máximo local. En definitiva, *gradient ascent* explora el espacio de variables latentes del modelo y se desplaza en la dirección de máxima pendiente (la que le indica el gradiente de la función de coste) hasta encontrar un montículo (máximo local). Los parámetros del modelo variacional se actualizan de la siguiente manera:

$$\lambda = \lambda - \rho \nabla_{\lambda} ELBO(\lambda)$$

Durante los últimos años han ido apareciendo optimizaciones de este algoritmo: *Momentum*, *Adagrad*, *Adadelta*, *RMSprop*, *Adam*, ... las mejoras de los cuales se basan en aspectos tales como que cada parámetro de la función de coste tenga su propio *learning rate* η_i o teniendo en cuenta el valor de los gradientes de iteraciones anteriores para calcular el siguiente [40].

3.5.2 Algoritmo

El algoritmo GAVI quedaría de la siguiente manera:

Algorithm 3: GAVI algorithm.

```
1 Initialize  $\lambda$  randomly
2 repeat
3    $\lambda = \lambda - \rho \nabla_{\lambda} ELBO(\lambda, X)$ 
4   Recalculate ELBO
5 until ELBO grows no more;
```

3.5.3 Ejemplo

En el siguiente ejemplo se muestra, mediante una función simple en un espacio de 2 dimensiones, como se desarrollaría una optimización mediante *gradient descent*. La función a optimizar es:

$$f(x, y) = 5x^2 - 6xy + 5y^2$$

Tan solo se muestra el proceso de una iteración del algoritmo. Es decir, la obtención de un vector gradiente. En la figura 3.9 se puede apreciar como va disminuyendo el *learning rate* (la distancia recorrida en la dirección del vector gradiente) a medida que se va acercando al mínimo relativo.

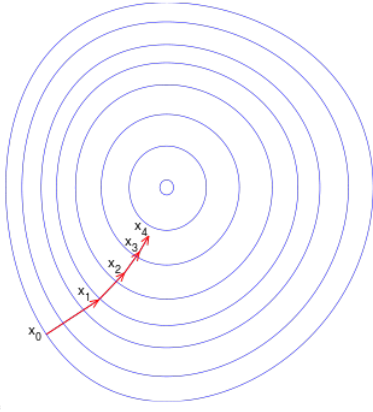


Figura 3.9: Representación gráfica de la búsqueda de un mínimo relativo realizada con *gradient descent* en un espacio de 2 dimensiones

- Se empieza en un punto aleatorio (x,y) del espacio: $(-0.5, 0.5)$
- Se obtiene $f(x)$ y $f(y)$

$$f(x) = 5x^2 - 3x + 1.25$$

$$f(y) = 5y^2 + 3y + 1.25$$

- Se deriva $f(x)$ y $f(y)$

$$\partial f(x) = 10x - 3$$

$$\partial f(y) = 10y + 3$$

- Se igualan las derivadas a 0 para obtener el vector gradiente

$$\nabla f(-0.5, 0.5) = (0.3, -0.3)$$

- Se actualiza el valor de x e y

$$(x, y) = (-0.5, 0.5) - \eta(0.3, -0.3)$$

3.5.4 Consideraciones

Un problema de este algoritmo para la aproximación del *posterior*, el cual provoca unas convergencias más inexactas, es la utilización del gradiente para la optimización de los parámetros variacionales. El gradiente supone que el espacio de variables latentes es un espacio Euclídeo. Este hecho implica la suposición de que la distancia entre las distribuciones se mide mediante la distancia Euclídea entre sus parámetros. La solución a este problema, encontrar la distancia real entre dos distribuciones, pasa por utilizar el gradiente natural.

$$\hat{\nabla}_{\lambda} ELBO(\lambda) \approx \mathbb{G}(\lambda)^{-1} \nabla_{\lambda} ELBO(\lambda)$$

El gradiente natural indica la dirección de máxima pendiente en un espacio, el espacio de *Riemman*, donde se tiene en cuenta la distancia real entre distribuciones y se puede calcular multiplicando el gradiente normal por la inversa de la matriz de *Fisher*, $\mathbb{G}(\lambda)$, la cual requiere el cálculo de la segunda derivada.

$$\mathbb{G}(\lambda) = \mathbb{E}_{\theta}[(\nabla_{\lambda} \ln(q(\theta|\lambda)))(\nabla_{\lambda} \ln(q(\theta|\lambda)))^T] = Cov_{\theta}[\nabla_{\lambda} \ln(q(\theta|\lambda))] = \mathbb{E}_{\theta}[-\nabla_{\lambda}^2 \ln(q(\theta|\lambda))]$$

En el caso de CAVI, cuando se derivan las actualizaciones analíticas de cada parámetro variacional, ya se está teniendo en cuenta la forma de las distribuciones para medir la distancia.

3.6 Problemas de eficiencia

Hoy en día, los algoritmos que se utilizan para el *machine learning* utilizan grandes volúmenes de datos. Esto provoca que los programadores escalen y distribuyan los algoritmos a través de diversas máquinas o diseñen alternativas simplificadas y menos costosas computacionalmente. En el caso de CAVI y GAVI se hace una pasada por todos los datos del *dataset* (*epoch*) en cada iteración del algoritmo. Este procedimiento, para *datasets* muy grandes, es impracticable. En los próximos apartados se mostrará como solventar esta problemática.

3.7 Stochastic Variational Inference

La existencia de grandes *datasets* es la principal motivación para la creación del algoritmo que se presenta a continuación. *Stochastic Variational Inference* (SVI) [41, 42] es una extensión de CAVI y GAVI que permite aumentar la escalabilidad de VI. Esta versión se basa principalmente en el uso de un *batch* (subconjunto de los datos) del *dataset* en cada iteración. De esta forma, tras realizar más iteraciones que las que realizaría con las versiones convencionales de VI, la solución irá tendiendo a un óptimo local. La principal ventaja de este mecanismo es que no requiere tener todo el *dataset* en memoria, resolviendo el cuello de botella que se podría formar al usar VI con *datasets* muy grandes.

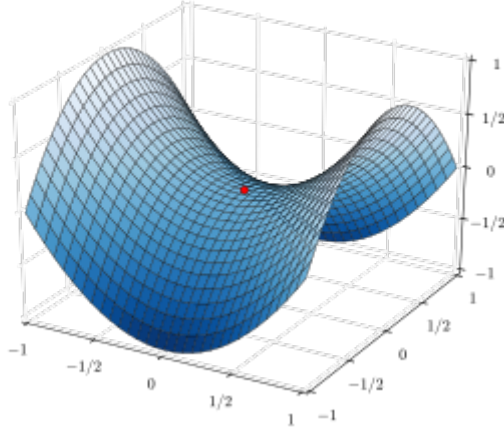


Figura 3.10: Saddle point (Fuente: Wikipedia)

3.7.1 Stochastic optimization

Se trata de la técnica que obtiene estimaciones del gradiente real de una función objetivo. De esta forma se obtiene un algoritmo que itera sobre cada punto o *batch* de forma aleatoria y va ajustando la estructura oculta del modelo basándose solo en ese punto o *batch* en cada iteración. *Stochastic optimization* encuentra un máximo o mínimo relativo de una función mediante estimaciones del gradiente real. La *expectation* de las estimaciones $\mathbb{E}(\nabla_{\lambda} ELBO(\lambda, x))$, siendo x un *batch* de datos, equivale al gradiente real $\nabla_{\lambda} ELBO(\lambda, X)$, siendo X el *dataset* completo.

$$\lambda = \lambda - \rho \nabla_{\lambda} ELBO(\lambda, x)$$

En las condiciones idóneas estos algoritmos estocásticos convergen a un óptimo local de la función siempre que ρ cumpla las condiciones de Robins-Monro [43]:

$$\sum_{t=0}^{\infty} \rho_t = \infty$$

$$\sum_{t=0}^{\infty} \rho_t^2 < \infty$$

La utilización de esta técnica da lugar al algoritmo *Stochastic Gradient Ascent Variational Inference* (SGAVI) y si se utilizan los gradientes naturales a *Stochastic Natural Gradient Ascent Variational Inference* (SNGAVI). Además, estos algoritmos, gracias a las estimaciones del gradiente real, permiten salir de los *saddle points* (véase la figura 3.10). Existen técnicas como la reparametrización del gradiente [44] que permiten reducir la varianza producida por el cálculo estocástico del gradiente.

3.7.2 Algoritmo

En el caso de VI la función a optimizar es la ELBO. Si los parámetros variacionales se actualizan mediante las fórmulas analíticas cerradas se conoce como *Stochastic Coordinate Ascent Variational Inference* (SCAVI) mientras que si utilizamos *stochastic optimization* se trata de SGAVI. Además, se añade un término corrector utilizando los cálculos de iteraciones anteriores. El algoritmo genérico quedaría de la siguiente manera:

Algorithm 4: SVI algorithm.

```

1 Initialize  $\lambda$  randomly
2 repeat
3   Sample a data batch of  $X$ 
4   Compute its local parameter
5   Update the current estimate of the global parameters
6    $\lambda^t = (1 - \rho_t)\lambda^{t-1} + \rho_t \hat{\lambda}$ 
7 until ELBO grows no more;
```

3.8 Black Box Variational Inference

Otra aproximación para el problema de VI es *Black Box Variational Inference* (BBVI). Consiste en realizar un muestreo del modelo variacional $q(\theta|\lambda)$ con el objetivo de realizar una aproximación de las *expectations* de la fórmula:

$$ELBO(q(\theta|\lambda), p(x, \theta)) = \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$$

Dichas *expectations* respecto al modelo variacional, pueden ser muy costosas de calcular computacionalmente y pueden suponer un cuello de botella a nivel de memoria del ordenador.

3.8.1 Score gradients

A continuación se aplican gradientes y se realizan una serie de transformaciones algebraicas sobre la fórmula analítica de la ELBO:

$$\begin{aligned} \nabla_{\lambda} ELBO(q(\theta|\lambda), p(x, \theta)) &= \nabla_{\lambda} \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda)) \\ &= \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \end{aligned}$$

Se puede ver la derivación completa en el apéndice A.1.6.

3.8.2 Monte-Carlo integration

Monte-Carlo integration [45] es un mecanismo para la aproximación de integrales. Consiste en muestrear la variable respecto a la que se integra y realizar un sumatorio del valor de la función dadas esas muestras. Cuantas más muestras se tomen de la variable más exacta será la aproximación. En nuestro caso se utilizará para aproximar la integral de la ELBO, la cual integra con respecto a la distribución $q(\theta|\lambda)$. Por ello, un conjunto de muestras θ_s obtenidas de la distribución $q(\theta|\lambda)$ permitirán la aproximación de la integral. La formulación quedaría de la siguiente manera:

$$\begin{aligned} \nabla_{\lambda} ELBO(q(\theta|\lambda), p(x, \theta)) &\approx \frac{1}{S} \sum_{s=1}^S [\ln p(x, \theta_s) - \ln q(\theta_s|\lambda)] \nabla_{\lambda} [\ln q(\theta_s|\lambda)] \\ \theta_s &\sim q(\theta|\lambda) \end{aligned}$$

Tras estas transformaciones se llega a que no es necesario sacar gradientes de toda la ELBO si no que basta con derivar el modelo variacional.

3.8.3 Algoritmo

A continuación se presenta el algoritmo de BBVI:

Algorithm 5: BBVI algorithm.

```

1 Input: data x, joint distribution p, mean field variational family
2 Initialize  $\lambda$  randomly,  $t=1$ 
3 repeat
4   x = sample a batch of data X
5   // Draw S samples from  $q(\theta|\lambda)$ 
6   for  $s=1$  in  $S$  do
7      $z[s] \sim q(\theta|\lambda)$ 
8    $\eta$  =  $t$ th value of Robbins Monro sequence
9    $\lambda = \lambda + \eta \frac{1}{S} \sum_{s=1}^S [\ln p(x, \theta_s) - \ln q(\theta_s|\lambda)] \nabla_{\lambda} [\ln q(\theta_s|\lambda)]$ 
10   $t = t + 1$ 
11 until ELBO grows no more;
```

En esta versión del algoritmo, además de muestrear el modelo variacional se muestrean los datos, esto se conoce como doble estocasticidad.

3.8.4 Consideraciones

Este algoritmo es el resultado de tomar una serie de medidas que pueden cuestionar la convergencia del algoritmo de VI. Desde suponer que $q(\theta|\lambda)$ factoriza (*mean field assumption*) hasta aproximar la integral de la ELBO mediante *Monte-Carlo integration*. Todo esto provoca que este algoritmo esté sometido a una elevada varianza, y dependiendo del modelo, a una convergencia lenta. Con el objetivo de reducir la varianza que provoca este método han aparecido mecanismos como:

- *Rao-Blackwellization*. El cual reduce la varianza de una variable aleatoria reemplazándola por su *expectation* condicional respecto a un subconjunto de variables.
- *Control variates*. Sustituye la *expectation* calculada mediante *Monte-Carlo integration* por otra función con la misma *expectation* pero con menos varianza.

No obstante, un aspecto positivo de este método es que no es necesario derivar las fórmulas analíticas de las actualizaciones de los parámetros variacionales ni la forma analítica de la ELBO. Esto permite el acceso a estos algoritmos a personas con conocimientos matemáticos más básicos. Lo único que se tiene que definir es la probabilidad conjunta (*joint probability*) del modelo $p(x, \theta)$ y el modelo variacional $q(\theta|\lambda)$. También existe otra aproximación, *Automatic Differentiation Variational Inference* (ADVI), que también posee estas ventajas y mejora la convergencia de modelos no conjugados que pueden suponer un reto para el resto de variantes de VI.

Antiguamente con VI solo se podían inferir modelos conjugados puesto que los modelos no conjugados no eran fácilmente derivables. La aparición de algoritmos como ADVI y BBVI permitieron la inferencia de este tipo de modelos ya que se pasaba de un cálculo analítico a una estrategia aproximativa.

3.9 Librerías de Variational Inference

La mayoría de algoritmos desarrollados en este proyecto se han implementado en *Python* y *Tensorflow*. No obstante, la curiosidad por implementar modelos probabilísticos nos llevó a probar algunas de las siguientes librerías, las cuales son las más conocidas para la inferencia de modelos probabilísticos con VI y MCMC.

- *Edward*⁴: Utiliza *Tensorflow* para el cómputo de gradientes y dispone de implementaciones para BBVI, *Reparameterization* BBVI, *Metropolis-Hastings*, ...
- *Stan*⁵: Utiliza una implementación en *C++* de *Automatic Differentiation* (AD) *reverse mode* [46] para el cómputo de gradientes y dispone de implementaciones para ADVI, HMC, ...
- *PyMC3*⁶: Utiliza *Theano* para el cómputo de gradientes y dispone de implementaciones para ADVI, *Gibbs Sampling*, *Metropolis-Hastings*, ...
- *BayesFlow*⁷: Módulo joven de *Google* para VI.

⁴ *Edward*: <https://github.com/blei-lab/edward>

⁵ *Stan*: <https://github.com/stan-dev/stan>

⁶ *PyMC3*: <https://github.com/pymc-devs/pymc3>

⁷ *BayesFlow*: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/bayesflow>

4. Gaussian Mixture Model

En este capítulo se explicará el modelo probabilístico con el que se ha trabajado principalmente en este proyecto: *Gaussian Mixture Model* (GMM). Se trata de un modelo de aprendizaje no supervisado. Este tipo de aprendizaje procura obtener una descripción de los datos. Concretamente, este modelo agrupa los datos modelando una serie de distribuciones que explicarán su generación. Cuando se habla de *mixtura* se hace referencia a un conjunto de distribuciones del mismo tipo que modelan subconjuntos de datos. En nuestro caso, se modela una *mixtura* de Gaussianas, es decir, una *mixtura* de distribuciones normales.

Se decidió trabajar con este modelo puesto que, al ser un modelo no conjugado, no es posible realizar *exact inference* para obtener el *posterior* y por tanto se requiere de una estrategia aproximativa como *Variational Inference* (VI) para su inferencia. En el capítulo de aplicaciones se mostrará la aplicabilidad de este modelo para agrupar rutas *Global Positioning System* (GPS).

4.1 Proceso generativo

GMM modela K distribuciones *Normal Inverse Wishart* (una para cada *cluster*), la proporción de puntos en cada *cluster* con una distribución *Dirichlet*, N distribuciones *Categorical* que definirán la pertenencia de cada punto a cada *cluster* y N distribuciones normales que establecerán la posible posición de cada punto.

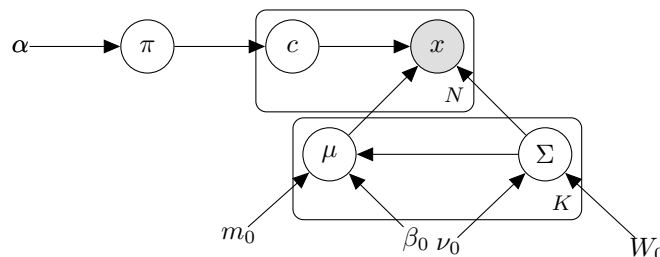
Como se ha comentado anteriormente, los modelos probabilísticos, al modelar la distribución de todas las variables, permiten definir un procedimiento generativo de nuevos datos utilizando los parámetros variacionales inferidos. A continuación se muestra dicho proceso para el modelo GMM:

1. $\pi \sim \text{Dir}(\lambda_\pi)$
2. $\mu_k, \Sigma_k \sim \text{NIW}(\lambda_{m_k}, \lambda_{\text{beta}_k}, \lambda_{\nu_k}, \lambda_{W_k})$ for $k \in \{1, \dots, K\}$
3. $C_n \sim \text{Cat}(\pi)$, $x_n \sim N(\mu_{C_n}, \Sigma_{C_n})$ for $n \in \{1, \dots, N\}$

Este proceso se resume en que para cada dato, primero se decide qué distribución lo ha generado y después se genera el dato. En la implementación de este algoritmo¹ únicamente deben suministrarse los parámetros variacionales inferidos mediante algún método de VI y el algoritmo se encarga de generar nuevos datos sacando muestras de las distribuciones del modelo.

4.2 Probabilistic graphical model y probabilidad conjunta

A continuación se muestra el *probabilistic graphical model* del modelo GMM:



¹Algoritmo generativo: https://github.com/bertini36/GMM/blob/master/inference/python/generate_new_data.py

Donde para cada una de las variables aleatorias del modelo se utilizaron las siguientes distribuciones:

- π : Una distribución *Dirichlet* con parámetro α ($Dir(\alpha)$). Se encarga de modelar la proporción de asignaciones de puntos a cada uno de los *clusters*.
- c : N distribuciones *Categorical* con parámetro el vector de proporciones modelado por la distribución *Dirichlet* ($Cat(\pi)$). Se encargan de modelar la probabilidad de pertenencia de cada uno de los puntos a cada *cluster*.
- μ y Σ : K distribuciones *Normal Inverse Wishart* con parámetros: m_0 , β_0 , ν_0 y W_0 . Esta distribución modela la media y la matriz de covarianzas de cada uno de los *clusters*.
- x : N distribuciones normales con parámetros: μ , Σ y c . Esta distribución modela la posición del punto en base a la media y varianza del *cluster* al que pertenece.

La definición de la probabilidad conjunta del modelo se escribiría de la siguiente manera:

$$p(\pi, c, \mu, \Sigma, x | \alpha, m_0, \beta_0, \nu_0, W_0) = p(\pi | \alpha) \prod_{k=1}^K p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1}^N p(c_n | \pi) p(x_n | \mu_{c_n}, \Sigma_{c_n}) \mathbf{I}(c_n = k)$$

Siendo π , c , μ , Σ y x las variables del modelo, α el *prior* de la distribución *Dirichlet* y m_0 , β_0 , ν_0 y W_0 los *priors* de la distribución *Normal Inverse Wishart*. Y si se aplica el logaritmo a la probabilidad conjunta queda como:

$$\ln p(\pi, c, \mu, \Sigma, x) = \ln p(\pi | \alpha) + \sum_{n=1}^N \ln p(c_n | \pi) + \sum_{k=1}^K \left(\ln p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) + \sum_{n=1, \mathbb{I}(c_n=k)}^N \ln p(x_n | \mu_k, \Sigma_k) \right)$$

4.3 Intratabilidad del posterior

Tal como se ha explicado anteriormente, la existencia de métodos para aproximar el *posterior* como *Markov Chain Monte Carlo* (MCMC) o VI se debe principalmente a la intratabilidad computacional de la distribución *posterior* en determinados modelos probabilísticos. Si existe una cantidad elevada de datos o el modelo tiene un número elevado de variables latentes, el cálculo de la evidencia en el denominador de la regla de Bayes es intratable.

Recordatorio de la fórmula del *posterior*/regla de Bayes:

$$p(\theta | x) = \frac{p(x | \theta) p(\theta)}{p(x)}$$

En el caso de GMM la formulación del *posterior* quedaría de la siguiente manera:

$$\begin{aligned} p(\pi, c, \mu, \Sigma | x, \alpha, m_0, \beta_0, \nu_0, W_0) &= \frac{p(\pi, c, \mu, \Sigma, x | \alpha, m_0, \beta_0, \nu_0, W_0)}{\sum_{c_{1:N}} \int_{\mu_{1:K}} \int_{\Sigma_{1:K}} p(\pi, c, \mu, \Sigma, x | \alpha, m_0, \beta_0, \nu_0, W_0)} = \\ &= \frac{Dir(\pi | \alpha) \prod_{k=1}^K NIW(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1}^N Cat(c_n | \pi) N(x_n | \mu_{c_n}, \Sigma_{c_n}) \mathbf{I}(c_n = k)}{\sum_{c_{1:N}} \int_{\mu_{1:K}} \int_{\Sigma_{1:K}} Dir(\pi | \alpha) \prod_{k=1}^K NIW(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1}^N Cat(c_n | \pi) N(x_n | \mu_{c_n}, \Sigma_{c_n}) \mathbf{I}(c_n = k)} \end{aligned}$$

El denominador de esta expresión (la evidencia del modelo), si se utilizan una cantidad significativa de datos para la inferencia, es intratable. Por esta razón, GMM es un buen modelo de ejemplo para inferir con métodos de VI.

4.4 Modelo variacional

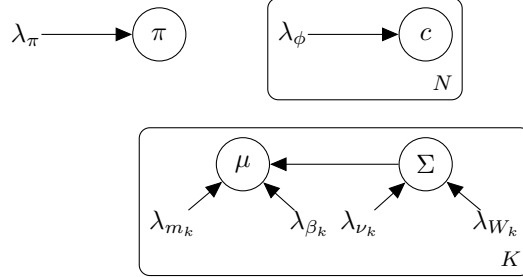
El modelo variacional nos permitirá aproximar el modelo probabilístico (el *posterior*). El objetivo es definir un modelo simplificado que mediante la optimización de sus parámetros nos permita aproximar lo mejor posible el modelo probabilístico. Para ello se suelen tomar medidas como eliminar las evidencias y algunas relaciones entre variables. Para nuestro proceso de VI se definió el siguiente modelo variacional:

$$q(\pi, c, \mu, \Sigma | \lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) = q(\pi | \lambda_\pi) \prod_{k=1}^K q(\mu_k, \Sigma_k | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) \prod_{n=1}^N q(c_n | \lambda_\phi)$$

Y si se aplican logaritmos queda como:

$$\ln q(\pi, c, \mu, \Sigma | \lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) = \ln q(\pi | \lambda_\pi) + \sum_{k=1}^K \ln q(\mu_k, \Sigma_k | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) + \sum_{n=1}^N \ln q(c_n | \lambda_\phi)$$

Con su respectivo *probabilistic graphical model*:



λ_π , λ_ϕ , λ_m , λ_β , λ_ν y λ_W son los parámetros variacionales que se inferirán mediante VI y tienen las siguientes dimensiones:

Variational parameter	Shape
λ_π	(K)
λ_m	(D, K)
λ_β	(K)
λ_ν	(K)
λ_W	(D, D, K)
λ_ϕ	(N, K)

Siendo N el número de puntos, K el número de *clusters* y D el número de dimensiones de cada punto.

4.5 Derivación de los parámetros variacionales

En este apartado se mostrarán las fórmula analíticas utilizadas para actualizar los parámetros variacionales en cada iteración del algoritmo y que se implementaron para el algoritmo de *Coordinate Ascent Variational Inference* (CAVI). Para la obtención de estas actualizaciones se utilizan las propiedades de la *Exponential Family* ya mencionadas en apartados anteriores. Las derivaciones completas de cada parámetro se pueden ver en A.2.3, A.2.4 y A.2.5.

$$\lambda_\pi = \begin{bmatrix} \alpha_1 + \sum_{n=1}^N \lambda_{\phi_{n,1}} \\ \vdots \\ \alpha_1 + \sum_{n=1}^N \lambda_{\phi_{n,k}} \end{bmatrix}$$

$$\lambda_m = \begin{bmatrix} \frac{m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,1}} x_n}{\lambda_{\beta_1}} \\ \vdots \\ \frac{m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n}{\lambda_{\beta_k}} \end{bmatrix}$$

$$\lambda_\beta = \begin{bmatrix} \beta_0 + N_1 \\ \vdots \\ \beta_0 + N_k \end{bmatrix}$$

$$\lambda_\nu = \begin{bmatrix} \nu_0 + N_1 \\ \vdots \\ \nu_0 + N_k \end{bmatrix}$$

$$\lambda_W = \begin{bmatrix} W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,1}} x_n x_n^T - \lambda_{\beta_1} \lambda_{m_1} \lambda_{m_1}^T \\ \vdots \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n x_n^T - \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \end{bmatrix}$$

$$\lambda_\phi = \text{Softmax} \left(\begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \lambda_{m_1} \lambda_{\nu_1} \lambda_{W_1}^{-1} x_n - \frac{1}{2} \lambda_{\nu_1} \lambda_{W_1}^{-1} x_n x_n^T \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \lambda_{m_k} \lambda_{\nu_k} \lambda_{W_k}^{-1} x_n - \frac{1}{2} \lambda_{\nu_k} \lambda_{W_k}^{-1} x_n x_n^T \\ - \frac{D}{2} \lambda_{\beta_1}^{-1} - \frac{\lambda_{\nu_1} \lambda_{m_1}^T \lambda_{W_1}^{-1} \lambda_{m_1}}{2} + \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_1}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_1}| \\ \vdots \\ - \frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \lambda_{m_k}^T \lambda_{W_k}^{-1} \lambda_{m_k}}{2} + \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} \right)$$

4.6 Model Evidence Lower Bound (ELBO)

En VI la condición de parada viene definida por la *Evidence Lower Bound* (ELBO). Para este modelo probabilístico la ELBO utilizada fue la siguiente:

$$\begin{aligned} ELBO(q(\theta|\lambda), p(x, \theta)) &= \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda)) = \\ &= - \sum_{i=1}^K \ln \Gamma(\alpha_i) + \ln \Gamma\left(\sum_{j=1}^K \alpha_j\right) - \frac{D(N+1)}{2} K \cdot \ln(2\pi) - K \cdot \frac{\nu_0 D}{2} \ln 2 - K \cdot \ln \Gamma_D\left(\frac{\nu_0}{2}\right) \\ &\quad + K \cdot \frac{D}{2} \ln |\beta_0| + K \cdot \frac{\nu_0}{2} \ln |W_0| + \sum_{i=1}^K \ln \Gamma(\lambda_{\pi_i}) - \ln \Gamma\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \frac{D}{2} K \cdot \ln(2\pi) + \frac{\lambda_{\nu_k} D}{2} \ln 2 \\ &\quad + \ln \Gamma_D\left(\frac{\lambda_{\nu_k}}{2}\right) - \frac{D}{2} \ln |\lambda_{\beta_k}| - \frac{\lambda_{\nu_k}}{2} \ln |\lambda_{W_k}| - \sum_{n=1}^N \begin{bmatrix} \ln \lambda_{\phi_{n,1}} \\ \vdots \\ \ln \lambda_{\phi_{n,k}} \end{bmatrix} \begin{bmatrix} \lambda_{\phi_{n,1}} \\ \vdots \\ \lambda_{\phi_{n,k}} \end{bmatrix} \end{aligned}$$

La derivación completa de esta ELBO se puede ver en el apéndice A.2.6. De esta última expresión es de la que se ha realizado la implementación para el algoritmo de CAVI². Cabe destacar que para las versiones basadas en gradientes (*Gradient Ascent Variational Inference* (GAVI) y *Stochastic Gradient Ascent Variational Inference* (SGAVI)) se utilizó otra ELBO basada en la del modelo *Warble*³ [47, 48] puesto que nuestra formulación no utilizaba el parámetro variacional λ_m y para el cálculo de gradientes se requería de una formulación de la ELBO que utilizara todos los parámetros variacionales.

4.7 Implementaciones

4.7.1 Coordinate Ascent Variational Inference

El primer algoritmo implementado para la inferencia del modelo GMM fue CAVI⁴. Su desarrollo requirió la implementación de las fórmulas derivadas para las actualizaciones de los parámetros variacionales y para la ELBO. La implementación utiliza librerías como *Numpy*⁵ y *Scipy*⁶.

²Código: https://github.com/bertini36/GMM/blob/master/inference/python/gmm_cavi.py

³*Warble*: <https://github.com/jcapde/WARBLE>

⁴GMM con CAVI: https://github.com/bertini36/GMM/blob/master/inference/python/gmm_cavi.py

⁵*Numpy*: <https://github.com/numpy/numpy>

⁶*Scipy*: <https://github.com/scipy/scipy>

El algoritmo quedó de la siguiente manera:

Algorithm 6: GMM inference with CAVI algorithm

```

1 repeat
2   Update  $\lambda_\pi$  with its formula
3   for  $n$  in  $N$  do
4     Update  $\lambda_{\phi_n}$  with its formula
5   for  $k$  in  $K$  do
6     Update  $\lambda_{m_k}$  with its formula
7     Update  $\lambda_{\beta_k}$  with its formula
8     Update  $\lambda_{\nu_k}$  with its formula
9     Update  $\lambda_{W_k}$  with its formula
10  Compute ELBO
11 until ELBO grows no more;

```

4.7.2 Gradient Ascent Variational Inference

Utilizando la ELBO del modelo *Warble* se realizó una implementación de GAVI⁷ mediante la librería de diferenciación automática *Tensorflow*. En esta versión la actualización de los parámetros variacionales globales se hace utilizando gradientes mientras que el parámetro local λ_ϕ se actualiza mediante la fórmula derivada para el algoritmo de CAVI.

El principal problema que surgió durante el desarrollo de este tipo de inferencia fue la mantenibilidad de la estabilidad numérica. Cuando se derivan fórmulas para todo, como es el caso de CAVI, las propias fórmulas aseguran las restricciones numéricas que tienen algunos de los parámetros variacionales del modelo. No obstante, el uso de gradientes para la optimización de los parámetros variacionales es un proceso mucho más exploratorio, y puede ocurrir que en una determinada iteración alguno de los parámetros tenga un valor restringido. Cuando ocurre esto, el cálculo de la ELBO mediante su fórmula analítica se ve comprometido por la aparición de valores *Not a Number* (NaN) que conllevan a errores en el algoritmo.

Las restricciones numéricas que tiene el modelo de GMM son las que siguen:

- Cada una de las K componentes del vector λ_π debe ser un número positivo.
- λ_β debe ser positivo.
- λ_ν debe ser positivo y mayor que la dimensionalidad de los datos.
- λ_W debe ser una matriz definida positiva.
- La suma de cada fila n de λ_ϕ debe sumar 1 ya que cada fila de la matriz representa un vector de proporciones de la pertenencia de un punto a cada uno de los *clusters*.

Tensorflow ofrece una serie de funciones para restringir numéricamente el valor de sus variables. En nuestro caso se utilizó la función *softplus*⁸ para asegurar que los parámetros λ_π , λ_β y λ_ν fueran positivos y la función *softmax*⁹ para asegurar que las filas de la matriz λ_ϕ sumaban 1. Por último, se realizó una implementación en *Tensorflow* para la definición de matrices positivas mediante la descomposición de *Cholesky*¹⁰.

⁷GMM con GAVI: https://github.com/bertini36/GMM/blob/master/inference/tensorflow/gmm_gavi.py

⁸*Softplus*: https://www.tensorflow.org/api_docs/python/tf/nn/softplus

⁹*Softmax*: https://www.tensorflow.org/api_docs/python/tf/nn/softmax

¹⁰Descomposición de *Cholesky*: https://en.wikipedia.org/wiki/Cholesky_decomposition

El algoritmo quedó de la siguiente manera:

Algorithm 7: GMM inference with GAVI algorithm

```

1 repeat
2   for  $n$  in  $N$  do
3     Update  $\lambda_{\phi_n}$  with its formula
4   Compute ELBO gradients
5    $\lambda_\pi = \lambda_\pi - \rho \nabla_{\lambda_\pi} ELBO(\lambda_\pi, X)$ 
6   for  $k$  in  $K$  do
7      $\lambda_{m_k} = \lambda_{m_k} - \rho \nabla_{\lambda_{m_k}} ELBO(\lambda_{m_k}, X)$ 
8      $\lambda_{\beta_k} = \lambda_{\beta_k} - \rho \nabla_{\lambda_{\beta_k}} ELBO(\lambda_{\beta_k}, X)$ 
9      $\lambda_{\nu_k} = \lambda_{\nu_k} - \rho \nabla_{\lambda_{\nu_k}} ELBO(\lambda_{\nu_k}, X)$ 
10     $\lambda_{W_k} = \lambda_{W_k} - \rho \nabla_{\lambda_{W_k}} ELBO(\lambda_{W_k}, X)$ 
11   Compute ELBO
12 until ELBO grows no more;

```

4.7.3 Stochastic Coordinate Ascent Variational Inference

Para *Stochastic Coordinate Ascent Variational Inference* (SCAVI)¹¹ se adaptó el algoritmo de CAVI para que usara un *batch* de datos en vez de usar todos los datos en cada iteración.

4.7.4 Stochastic Gradient Ascent Variational Inference

Para SGAVI¹² se adaptó el algoritmo de GAVI para que usara un *batch* de datos en vez de usar todos los datos en cada iteración.

4.8 Otros modelos implementados

La realización del modelo GMM requirió de un conjunto de pasos intermedios que supusieron la implementación de modelos más simples. A continuación se nombran algunos de los modelos implementados en diversas tecnologías:

- *Dirichlet - Categorical model*. Disponible versión *Python* y versión *Edward*.
- *Inverse Gamma - Normal model*. Disponible versión *Python* y versión *Edward*.
- *Normal - Inverse Wishart model*. Disponible versión *Python*.
- *Univariate Gaussian*. Disponible versión *Python* y versión *Tensorflow*.

Todos estos modelos, incluido el modelo GMM, se pueden encontrar en las carpetas *models*¹³ y *inference*¹⁴ del repositorio *Github* del proyecto¹⁵.

¹¹GMM con SCAVI: https://github.com/bertini36/GMM/blob/master/inference/python/gmm_scavi.py

¹²GMM con SGAVI: https://github.com/bertini36/GMM/blob/master/inference/tensorflow/gmm_sgavi.py

¹³Otros modelos: <https://github.com/bertini36/GMM/blob/master/models>

¹⁴*Univariate Gaussian Model* (UGM) con CAVI y GAVI: <https://github.com/bertini36/GMM/blob/master/inference>

¹⁵Repositorio del proyecto: <https://github.com/bertini36/GMM>

5. Experimentación

En este capítulo se presentan diversas comparativas entre los diferentes métodos de *Variational Inference* (VI) implementados. Se comparan las versiones basadas en formulación analítica: *Coordinate Ascent Variational Inference* (CAVI) y *Stochastic Coordinate Ascent Variational Inference* (SCAVI); frente a las basadas en gradientes: *Gradient Ascent Variational Inference* (GAVI) y *Stochastic Gradient Ascent Variational Inference* (SGAVI). También se verán las ventajas que ofrecen las versiones estocásticas con respecto a las clásicas y viceversa.

Todas las pruebas hechas para este capítulo se han realizado con *datasets* sintéticos de 100 o 1000 puntos de 2 dimensiones, preparados para diferentes configuraciones de *clusters* y con un tamaño de *batch* de 100 para los algoritmos estocásticos.

5.1 Comparación de ELBOs

En este apartado se comparan las tendencias que produce la *Evidence Lower Bound* (ELBO) con las implementaciones de CAVI, GAVI, SCAVI y SGAVI. En las figuras 5.1a y 5.2a está representada la tendencia de la ELBO en los diferentes algoritmos para un caso de $K = 2$ y para un caso de $K = 4$. En las figuras 5.1b y 5.2b se han normalizado los valores mediante la función logarítmica con el objetivo de poder apreciar mejor las diferencias entre las diferentes tendencias. Como se ha comentado en diversas ocasiones la ELBO

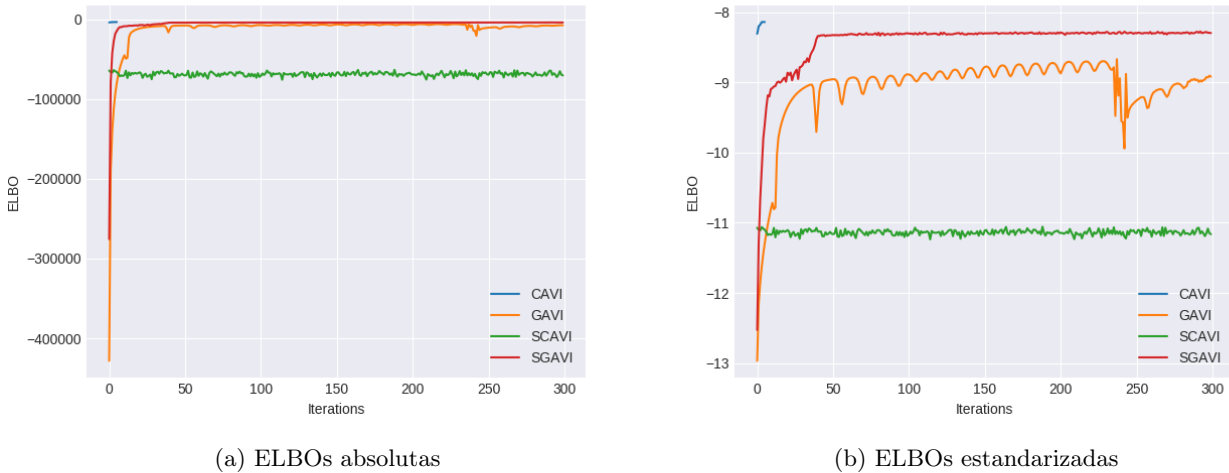


Figura 5.1: Ejecución con $K = 2$ y limitado a 300 iteraciones

es una medida que describe la calidad del modelo. Se trata de una función estrictamente creciente que cuanto más cercana esté del valor 0 mejor son las distribuciones obtenidas para aproximar el *posterior*. Hay que tener en cuenta que si las actualizaciones de los parámetros variacionales se hacen mediante gradientes o si se usan versiones estocásticas, la ELBO deja de ser estrictamente creciente para ser simplemente creciente en tendencia, tal como vemos en las gráficas. Como se puede ver en las figuras 5.1 y 5.2, independientemente del algoritmo, la función siempre tiende a 0. No obstante, se puede ver como CAVI (representado en color azul), al obtener el valor de los parámetros variacionales de forma analítica, obtiene la mejor convergencia de la ELBO y en un número de iteraciones mucho menor al resto.

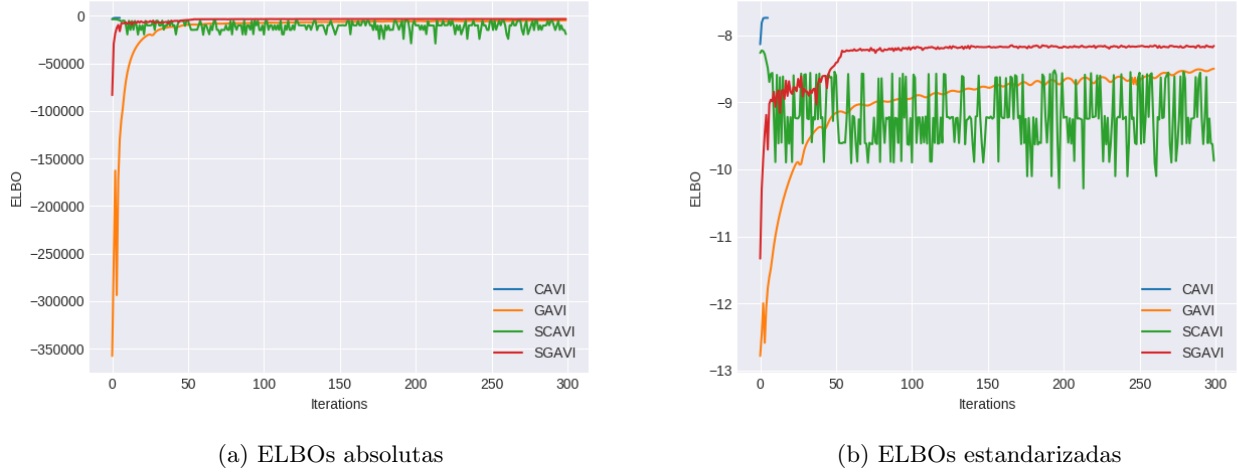


Figura 5.2: Ejecución con $K = 4$ y limitado a 300 iteraciones

Otra cosa que se puede ver en las gráficas es la superioridad en el nivel de convergencia de los algoritmos no estocásticos. Tanto CAVI como GAVI (representados en los colores azul y rojo respectivamente) llegan a cotas de la ELBO un poco más altas que sus versiones estocásticas, cosa que parece lógica puesto que estos utilizan todo el *dataset* en cada iteración para la inferencia del modelo variacional.

También destaca la elevada varianza en los valores de la ELBO obtenidos por SCAVI (representado en color verde), a pesar de que desde el principio se mueva en unos valores de convergencia relativamente buenos. Aumentando el tamaño del *batch* se consigue reducir esta varianza.

5.2 Comparación del consumo de memoria

A continuación se compara el uso de memoria entre CAVI y GAVI. Para medir el consumo de memoria se utilizó la aplicación *Memory Profiler*¹. En la figura 5.3 se compara el consumo por parte del algoritmo de CAVI y el de GAVI para *datasets* de 100 y 1000 puntos.

Tal como se puede apreciar la solución con gradientes es ampliamente más costosa a nivel de memoria que la opción analítica. Esto es debido al uso de *Tensorflow*. Esta librería, antes de la ejecución de un programa, construye un grafo de flujo donde los nodos representan arrays multidimensionales, conocidos como tensores, y las aristas son operaciones matemáticas. Cuando se define una función en *Tensorflow*, en este caso la ELBO, se construye un grafo que ya reserva la memoria necesaria. La ELBO, al tener un bucle que itera por cada punto del *dataset*, construye un grafo con una rama individual por cada punto, lo que provoca la generación de un grafo bastante grande y en consecuencia, un uso excesivo de la memoria.

Este problema se soluciona con la versión estocástica de GAVI: SGAVI. Esta implementación, al solo usar un *batch* de los datos en cada iteración, genera un grafo de flujo más pequeño, reduciendo el uso de memoria y consecuentemente, el tiempo de su construcción. Cabe decir que *Tensorflow* está pensado desde un principio para este tipo de inferencia estocástica, por esa razón, la aproximación de GAVI con esta librería no es del todo acertada.

5.3 Comparación de tiempos

Para comparar tiempos se decidió utilizar el tiempo por iteración por diversas razones. En primer lugar, CAVI llega a unos valores de la ELBO superiores a los de cualquier otro algoritmo en pocas iteraciones. En segundo lugar, solo CAVI llega a valores de la ELBO lo suficientemente estacionarios como para usar su mejora como condición de parada del algoritmo. Tanto en GAVI, como en SCAVI y SGAVI se debe limitar el número de iteraciones. Por último, tanto GAVI como SGAVI, al utilizar la librería *Tensorflow*, pierden bastante tiempo

¹ *Memory Profiler*: https://pypi.python.org/pypi/memory_profiler

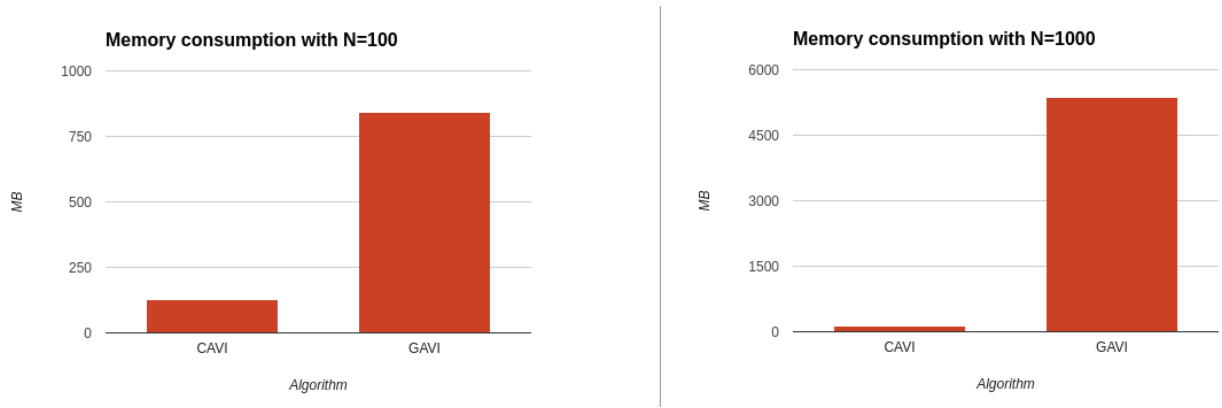


Figura 5.3: Comparación del consumo de memoria entre CAVI y GAVI

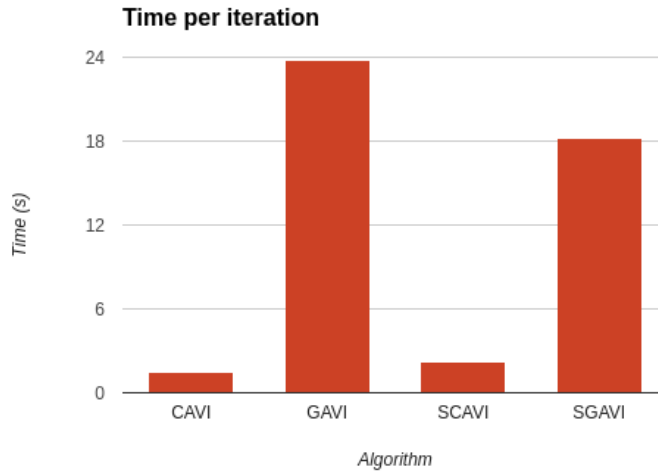


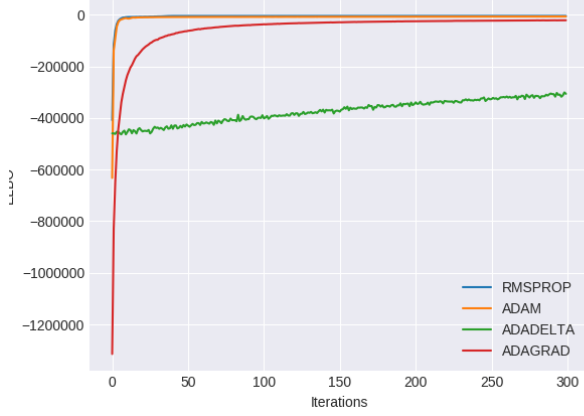
Figura 5.4: Ejecución con $K = 2$ y limitado a 300 iteraciones

al inicio de la ejecución para la construcción del grafo de flujo. En la figura 5.4 se puede apreciar una clara diferencia entre el tiempo por iteración en los algoritmos analíticos y el de los algoritmos basados en gradientes. Esto es debido a que derivar una función compleja, como es la ELBO, en cada iteración para obtener las actualizaciones de los parámetros variacionales es un proceso costoso.

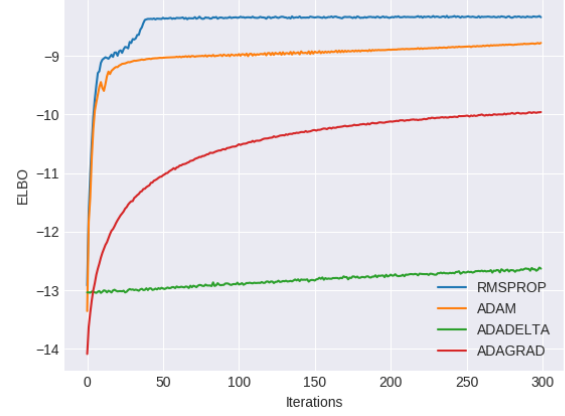
5.4 Comparación entre optimizadores

En este apartado se presentan los resultados de las ELBOs obtenidas por el algoritmo de SGAVI utilizando diferentes optimizadores que ofrece la librería *Tensorflow* para la optimización de variables mediante gradientes. Tal como se ha comentado anteriormente, existen variantes del algoritmo de *gradient descent* que mejoran su funcionamiento habitual. En las figuras 5.5 y 5.6 se comparan los optimizadores *Adagrad*, *Adadelata*, *RMSprop* y *Adam* con el objetivo de descubrir el más rápido en convergencia para el modelo de *Gaussian Mixture Model* (GMM). Tal como se puede ver en la figura 5.5b, parece que *RMSprop* (representado en color azul) es el que mejor se comporta en la inferencia de este modelo y que, en contraposición, *Adadelata* (representado en color verde) es el que peor resultados obtiene.

RMSprop [49] nace, al igual que *Adadelata*, para solventar el problema principal de *Adagrad*. *Adagrad* añade *learning rates* adaptativos al algoritmo de *gradient descent* convencional. Esto quiere decir que cada variable optimizable del modelo tendrá un *learning rate* η_i propio e irá adaptándose en función del óptimo de esa variable i . El problema de *Adagrad* es que estos *learning rates* particulares tienden rápidamente a 0 produciendo que la mejora obtenida por el algoritmo en cada iteración vaya disminuyendo drásticamente.

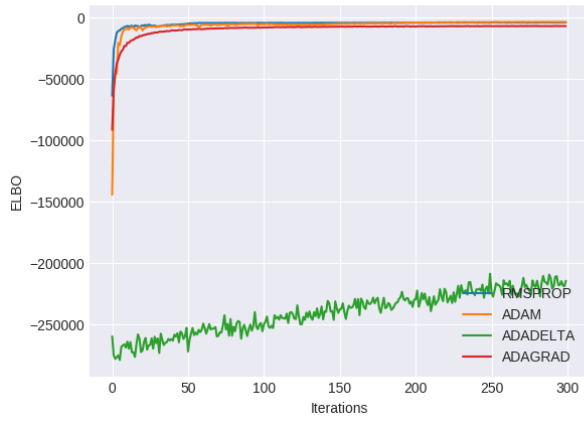


(a) ELBOs absolutas

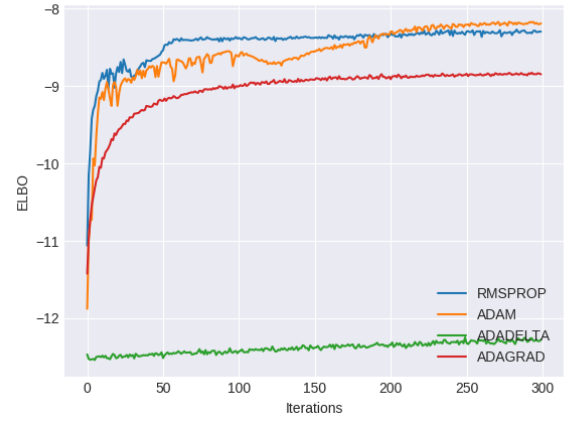


(b) ELBOs estandarizadas

Figura 5.5: Ejecución con $K = 2$, un *learning rate* inicial de 0.1 y limitado a 300 iteraciones



(a) ELBOs absolutas



(b) ELBOs estandarizadas

Figura 5.6: Ejecución con $K = 4$, un *learning rate* inicial de 0.1 y limitado a 300 iteraciones

Para solventar este problema *RMSprop* utiliza la media de los gradientes calculados anteriormente al cuadrado $\mathbb{E}[g^2]$. Esta media se calcula de forma recursiva en cada iteración t y se le da un peso γ al gradiente de la iteración actual.

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1 - \gamma) g_t^2$$

La configuración habitual de γ en este algoritmo es 0.9. Por último, la actualización de la variable se calcula como:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} g_t$$

5.5 Mejoras por hardware

Uno de los aspectos positivos de la librería *Tensorflow* es que compila al lenguaje *Compute Unified Device Architecture* (CUDA) [50]. CUDA es la plataforma que ofrece *Nvidia* para sacar el máximo provecho de sus *Graphic Processing Unit* (GPU) [51]. Con unos pequeños cambios en el código de SGAVI se puede especificar qué operaciones se requiere ejecutar en la GPU con el objetivo de aumentar el rendimiento. En este caso, la operación más costosa es el cálculo de los gradientes para la optimización de los parámetros variacionales, es decir, la derivación de la función ELBO en cada iteración. Así que en esta versión de SGAVI ² se especifica que dicha derivación se haga en la GPU.

²GPU SGAVI: https://github.com/bertini36/GMM/blob/master/inference/tensorflow/gmm_sgavi_minotau.py

En la figura 5.7 se pueden ver los resultados obtenidos, en términos de tiempo, al ejecutar el algoritmo de SGAVI en una *Central Processing Unit* (CPU) y en una GPU. Es un resultado bastante sorprendente. Lo que se esperaba era mejorar el tiempo con el uso de una GPU pero no fue así. Tras investigar los posibles motivos de que no se diera esta mejora del tiempo nos dimos cuenta de que *Tensorflow* no dispone de implementaciones para GPU de todas sus funciones, como por ejemplo:

- El gradiente del determinante de una matriz. El problema aquí reside en que este gradiente requiere sacar la inversa de la matriz ya que

$$\frac{\partial |X|}{\partial X} = |X| \cdot (X^{-1})^T$$

y la función `tf.matrix_inverse()` de *Tensorflow* no dispone de soporte para GPU.

- El gradiente de la función *Digamma*, necesaria para el cálculo de la *expectation* de la distribución *Dirichlet*, tampoco tiene soporte para GPU.

Principalmente dispone de las funciones más populares para lo que ha sido diseñado: el *deep learning*. La ELBO es una función compleja que utiliza funciones de *Tensorflow* poco utilizadas y sin soporte para la derivación con GPU. Esto provoca que no se pueda derivar íntegramente en la GPU y que se tenga que dividir su cómputo entre CPU y GPU. Esta división del cómputo de la derivación provoca que la CPU y la GPU se tengan que enviar resultados parciales constantemente, y es esta sobrecarga de mensajes la que provoca un tiempo peor que la versión que solo utiliza la CPU.

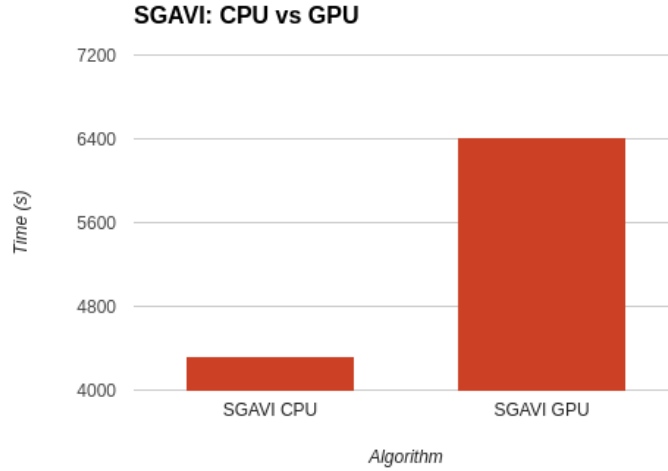


Figura 5.7: Comparación de tiempos en la realización de 300 iteraciones de la versión de SGAVI con CPU y la versión de SGAVI con GPU para $K = 2$

6. Aplicaciones

En este capítulo se mostrarán los resultados del modelo *Gaussian Mixture Model* (GMM) inferido con diferentes *datasets* (tanto sintéticos como reales) utilizando *Coordinate Ascent Variational Inference* (CAVI).

6.1 Datasets

6.1.1 Datasets sintéticos

Para la generación de datos sintéticos se implementaron *scripts* que, utilizando la implementación de distribuciones en librerías como *Numpy* y *Scipy*, modelaban unos datos de forma agrupada. De esta forma se realizaron pruebas con *datasets* de 1, 2, 3, 5 y hasta 7 dimensiones distribuidos en 2, 4, 8 o 30 *clusters*.

6.1.2 Dataset de Mallorca

Este *dataset* se obtuvo durante el desarrollo de mi Trabajo de Fin de Grado (TFG) mediante la realización de un *web crawler* que atacaba la web de *Wikiloc*¹. *Wikiloc* es un *sport tracker* donde los usuarios comparten sus trayectorias *Global Positioning System* (GPS) de diferentes actividades deportivas. Para evitar que la página detectara el procedimiento de obtención de rutas GPS como un ataque *Distributed Denial of Service* (DDoS) se utilizaron servidores *proxy*. Tras la depuración de los datos el resultado fue un conjunto de 1542 trayectorias

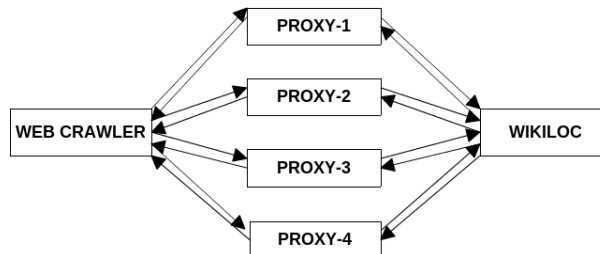


Figura 6.1: Obtención de las rutas GPS mediante servidores *proxy*

GPS en formato *GPS Exchange Format* (GPX) de la isla de Mallorca. Estas rutas han sido obtenidas por los dispositivos móviles de los usuarios de *Wikiloc*, por esta razón son rutas con un número elevado de puntos ya que estos dispositivos móviles suelen registrar la posición del usuario con bastante frecuencia. La media de puntos por trayectoria es de 1469. El formato de datos geográficos GPX, además de disponer de la latitud y longitud de los puntos, también dispone de la elevación y de una marca temporal, lo que nos permitió realizar agrupaciones de datos más específicas. En la figura 6.2 se muestra gráficamente la información de este *dataset*.

6.1.3 Dataset de Porto

Este *dataset* se obtuvo de la plataforma *Kaggle*². El conjunto está formado por 1.7 millones de trayectorias GPS generadas por los 442 taxis que circulan por la ciudad de Porto durante un año. Los dispositivos GPS de los taxis registran puntos en intervalos de 15 segundos lo que provoca que la media de puntos por ruta sea mucho menor que la del *dataset* de Mallorca: de unos 50 puntos por ruta. En la figura 6.3 se muestra gráficamente la información de este *dataset*.

¹ *Wikiloc*: <https://www.wikiloc.com/>

² *Kaggle*: <https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i>

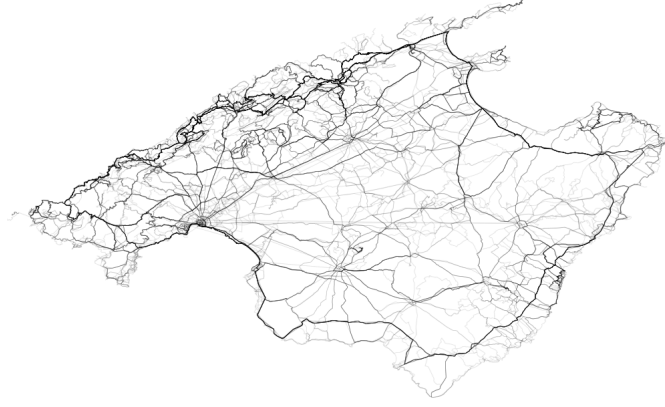


Figura 6.2: Representación gráfica del *dataset* de Mallorca

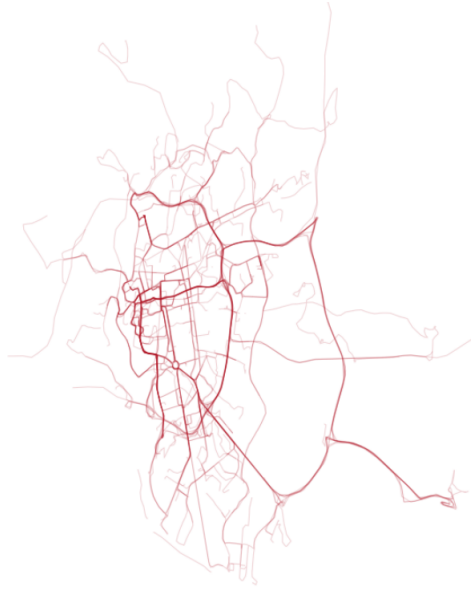


Figura 6.3: Representación gráfica del *dataset* de Porto

6.2 Preprocessing

En los siguientes apartados se narrarán las diferentes estrategias que se utilizaron para adecuar los datos reales a las necesidades de los algoritmos.

6.2.1 Interpolación de los datos

Como ya se ha explicado en apartados anteriores, GMM es un modelo probabilístico no supervisado que toma unos datos de entrada, puntos en un espacio de D dimensiones; y un número K de *clusters* para inferir diferentes variables aleatorias:

- La proporción de puntos a cada *cluster*: λ_π .
- La probabilidad de pertenencia de cada punto a cada *cluster*: λ_ϕ .
- La media y la matriz de covarianzas de cada *cluster*: λ_m y λ_W .

Dada esta definición práctica del modelo se deben interpolar los datos para que todos los puntos que forman el conjunto de datos tengan la misma dimensionalidad. Para la interpolación de las rutas de ambos *datasets* se hicieron pruebas con dos tipos de interpolaciones: la interpolación por vecinos cercanos y la interpolación lineal.

Interpolación por vecinos cercanos

Con el objetivo de estandarizar la dimensionalidad de los puntos a D dimensiones se divide la distancia (en número de puntos), entre el primer y último punto de la trayectoria, en N partes. Luego, para cada punto de la recta entre el primer y último punto, se selecciona el punto más cercano por posición en la ruta original para formar parte de la nueva ruta³. Esta aproximación se corresponde con la figura 6.4a.

Interpolación lineal

Mediante la ecuación de las rectas entre cada par de puntos y un segmento de longitud fija obtenido dividiendo la longitud de la ruta (distancia recorrida por la trayectoria) entre el número de puntos a los que se quiere interpolar, la interpolación lineal va recorriendo el camino formado por estas rectas y añadiendo los puntos que cortan con este segmento a la nueva ruta. Esta interpolación se hizo mediante la función *approx* de R^4 y se corresponde con la figura 6.4b.

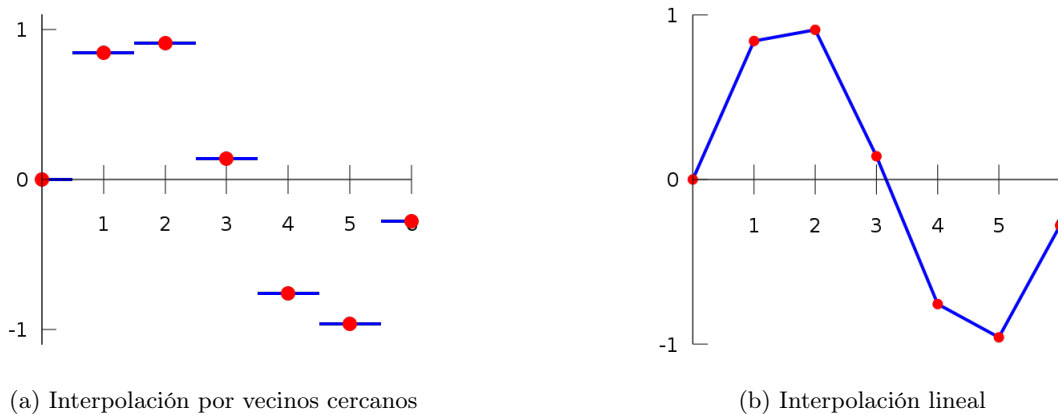


Figura 6.4: Procesos para la interpolación de rutas (Fuente: Wikipedia)

6.2.2 Reducción de dimensionalidad

Tras estandarizar la dimensionalidad de todas las rutas fue necesario un proceso de reducción de dimensionalidad para que el *dataset* completo cupiera en memoria durante el proceso de inferencia. Para este fin se valoraron 3 alternativas: *Principal Component Analysis* (PCA) [52], un *autoencoder* [53] y *Probabilistic Principal Component Analysis* (PPCA) [54]; con el interés de comprobar que el método de reducción no condicionara demasiado los resultados del algoritmo.

PCA

En términos comunes, este procedimiento encuentra el mejor punto desde el que mirar la nube de puntos. El concepto se basa en que no es lo mismo mirar una copa desde arriba que mirarla desde un lado. Mirándola desde un lado se podrá distinguir que se trata de una copa mientras que mirándola desde arriba no.

Este método busca la mejor proyección de los datos en términos de mínimos cuadrados. Realiza transformaciones lineales a los datos con el objetivo de descubrir un conjunto de vectores propios (componentes principales) que resultan ser los ejes que mejor representan los datos, es decir, los que recogen la mayor parte de la varianza. El principal problema de este método es que la elección del número de componentes principales que finalmente representarán los datos (la nueva dimensionalidad) la tiene que decidir el usuario. En otras palabras, el usuario debe establecer un umbral del porcentaje de varianza que quiere que explique la nueva proyección de los datos. Para su implementación⁵ se utilizó la función PCA de la librería *Python Sklearn*⁶.

³Interpolación NN: https://github.com/bertini36/GMM/blob/master/preprocessing/interpolation/nn_interpolation.py

⁴Interpolación lineal: https://github.com/jcapde/GMM/blob/master/preprocessing/interpolation/linear_interpolation.R

⁵PCA: <https://github.com/bertini36/GMM/blob/master/preprocessing/dimReduction/pca.py>

⁶*Sklearn*: <https://github.com/scikit-learn/scikit-learn>

Autoencoder

Un *autoencoder* es una red neuronal que se entrena para que aprenda representaciones comprimidas de los datos. Una red neuronal es un método de *machine learning* supervisado, es decir, que para su fase de entrenamiento se le suministra el resultado (*target*) que debe obtener para cada *input*. La red va modelando una función no lineal que permitirá inferir el *target* en datos nuevos.

En el caso de un *autoencoder* lo que se hace es entrenar la red para que prediga el propio *input*. En otras palabras, el *input* es el propio *target*. De esta manera, gracias a las capas ocultas (*hidden layers*), de menor dimensionalidad que el *input*, se obtendrán representaciones de los datos de menor dimensionalidad. La red neuronal que forma un *autoencoder* esta formada por un primer módulo, el *encoder* que dado el *input* obtiene una representación de menor dimensionalidad y otro módulo, el *decoder*, que dada esta representación comprimida del *input* es capaz de obtener nuevamente el *input*. En definitiva, lo que se hace es minimizar la diferencia entre el *input* y la salida del *decoder*.

La principal diferencia entre este método y PCA reside en que las transformaciones que realiza un *autoencoder*

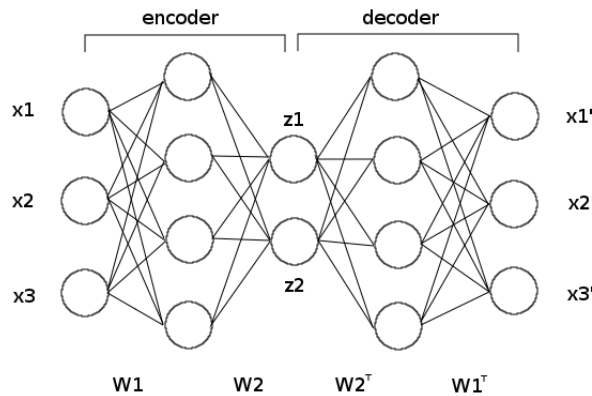


Figura 6.5: Estructura de un *autoencoder*. En esta figura, z (2 dimensiones) es la representación comprimida del dato x (3 dimensiones). La diferencia entre x y x' representa la pérdida del método de compresión

sobre los datos son no lineales, mientras que PCA solo realiza transformaciones lineales. Esto, dependiendo del *dataset*, puede producir proyecciones de los datos más complejas y ricas. No obstante, tiene el mismo problema que PCA puesto que se debe saber *a priori* la dimensionalidad de la proyección que se desea y configurar las capas ocultas de la red neuronal en consecuencia. Para su implementación⁷ se utilizó la librería *Keras*⁸.

PPCA

Con el objetivo de intentar obtener unos resultados similares a los del *paper* [32] también se utilizó PPCA para la reducción de dimensionalidad. Para ello se utilizó una implementación de este modelo probabilístico⁹ que utiliza la librería *Edward* y que además añade un coeficiente matemático para la evaluación del número de componentes principales a tener en cuenta: *Automatic Relevance Determination* (ARD). La especificación de este modelo probabilístico es la siguiente:

$$\begin{aligned}\alpha &\sim \text{Gamma}(1, 1) \\ w_{ij} &\sim \mathcal{N}(0, \alpha_j) \\ z_j &\sim \mathcal{N}(0, 1) \\ \mu_i &\sim \mathcal{N}(0, 1) \\ \sigma &\sim \text{Gamma}(1, 1) \\ x_i &\sim \mathcal{N}((w \cdot z)_i + \mu_i, \sigma)\end{aligned}$$

Tras realizar numerosas pruebas combinando estos métodos de reducción de dimensionalidad no se obtuvo una gran diferencia en la forma de agrupar los datos por parte del algoritmo.

⁷ *Autoencoder*: <https://github.com/bertini36/GMM/blob/master/preprocessing/dimReduction/ae.py>

⁸ *Keras*: <https://github.com/fchollet/keras>

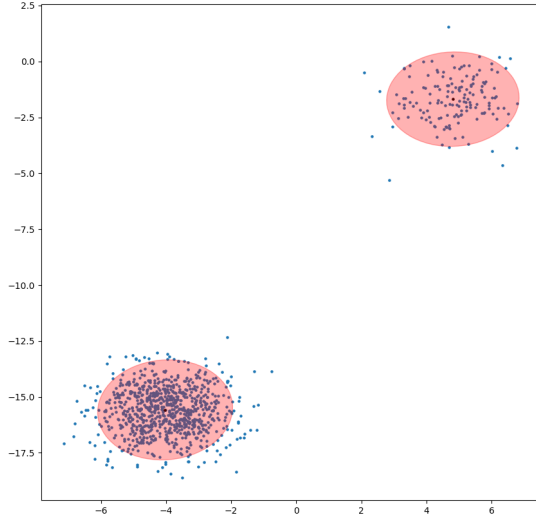
⁹ *PPCA*: <https://github.com/bertini36/GMM/blob/master/preprocessing/dimReduction/ppca.py>

6.3 Resultados

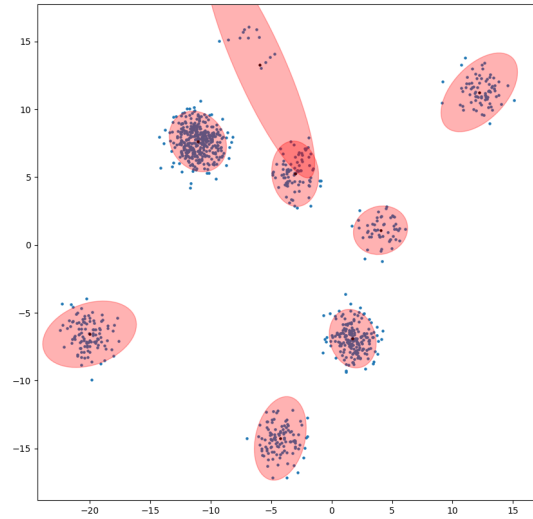
Todos los resultados que se muestran a continuación se obtuvieron mediante diferentes ejecuciones del algoritmo de CAVI.

6.3.1 Resultados con datasets sintéticos

Para estos *datasets* generados no se utilizó ninguno de los métodos de *preprocessing* nombrados anteriormente. Como se puede observar en las figuras (6.6a, 6.6b, 6.7a y 6.7b), el algoritmo modela una distribución

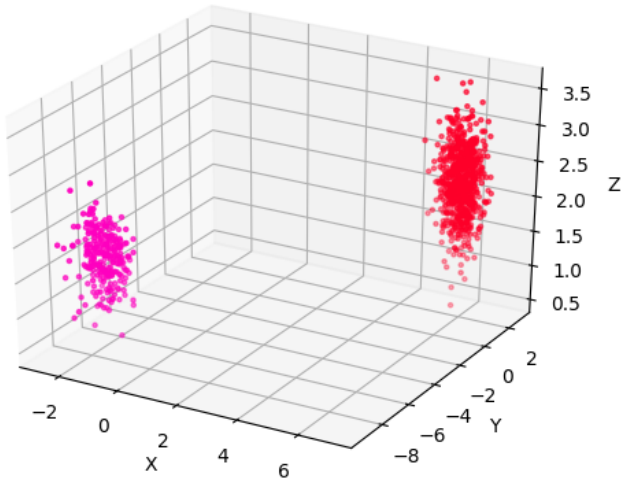


(a) Dataset de 2 clusters

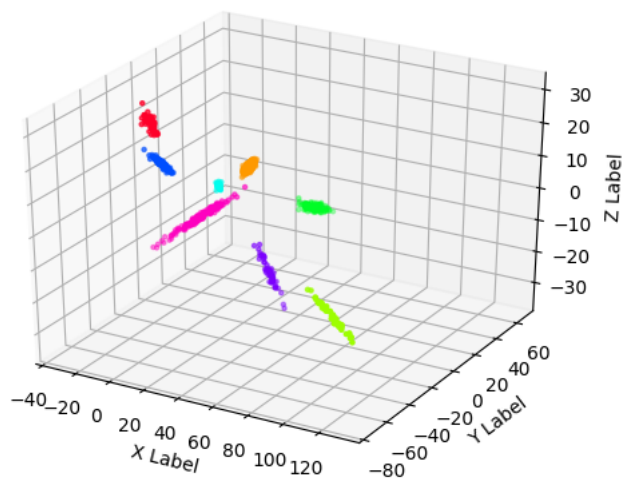


(b) Dataset de 8 clusters

Figura 6.6: Resultados del algoritmo con *datasets* sintéticos de 1000 puntos de 2 dimensiones



(a) Dataset de 2 clusters



(b) Dataset de 8 clusters

Figura 6.7: Resultados del algoritmo con *datasets* sintéticos de 1000 puntos de 3 dimensiones

Normal bastante acertada para cada *cluster*. A diferencia de otros algoritmos no supervisados de *clustering*, gracias a la inferencia de estas distribuciones se pueden generar nuevos datos para cada *cluster*, tal como se verá en un apartado posterior. Además, usando las proporciones inferidas por la distribución *Dirichlet* del modelo GMM, se puede incrementar el tamaño del *dataset* de una forma parecida a como lo haría en la realidad.

6.3.2 Resultados con el dataset de Mallorca

Para este *dataset* se utilizaron las diferentes estrategias de interpolación y reducción de dimensionalidad mencionadas en el apartado de *preprocessing* con el objetivo de obtener los mejores resultados posibles con el modelo GMM. Al final todo fue conseguir un equilibrio entre el número de puntos a los que interpolar las trayectorias y el tiempo de cómputo. Por un lado, si las trayectorias se interpolan a un número elevado de puntos (por ejemplo, la media) los resultados que se obtienen son mucho más ricos, pero se tarda demasiado en obtenerlos. Por otro lado, si se interpolan las rutas a un número reducido de puntos, los *clusters* que descubre el algoritmo son más básicos pero el tiempo de cómputo es mucho menor.

En las figuras 6.8, 6.9, 6.11 y 6.12 se muestran algunos de los resultados obtenidos utilizando diferentes métodos de reducción de dimensionalidad. Para la generación de los mapas¹⁰ se utilizó el lenguaje *R* y la librería *plotKML*.

Utilizando PCA para $K = 2$ se descubre un primer *cluster* (6.8a) que, como se puede apreciar en las

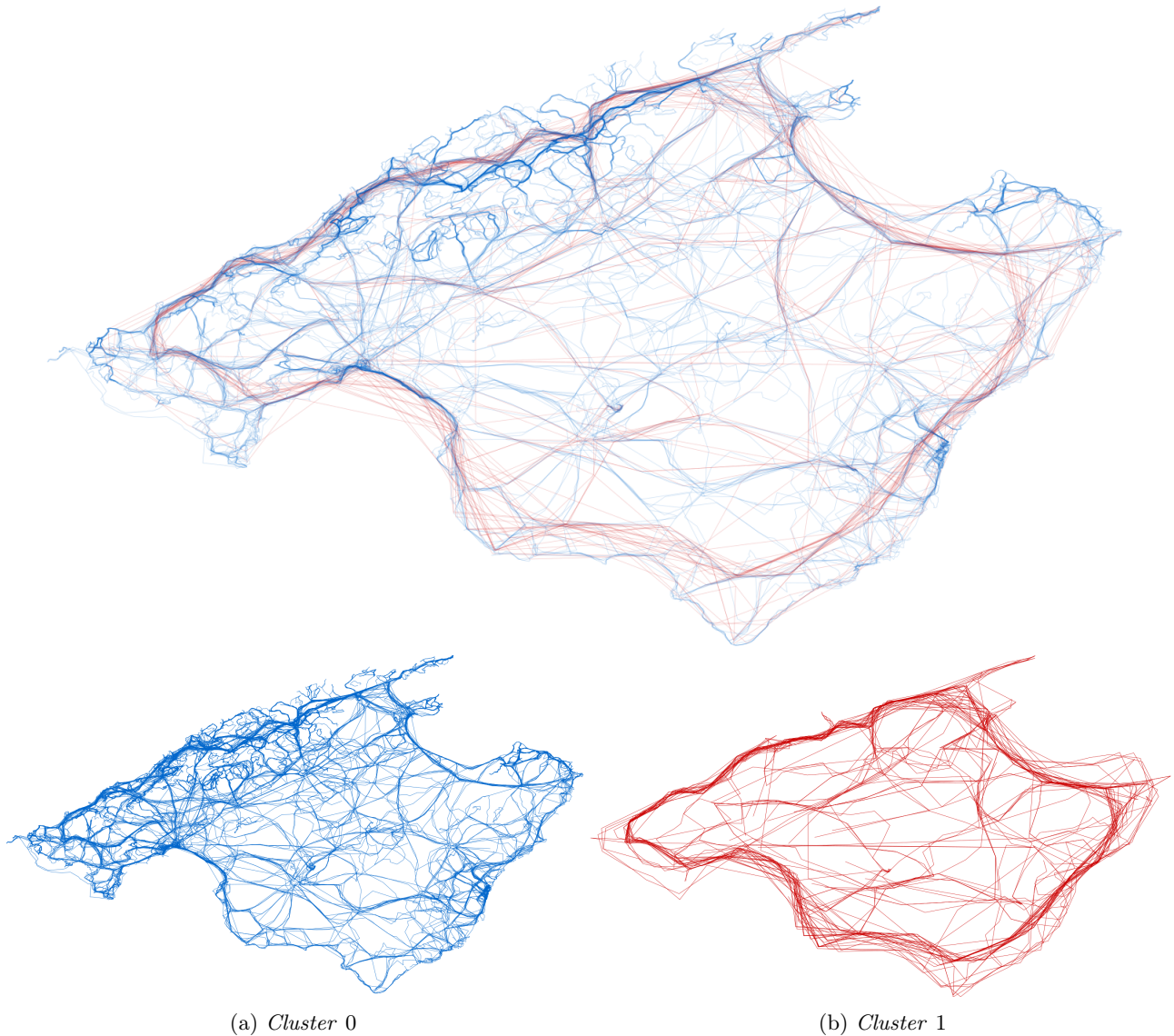


Figura 6.8: Resultados para $K = 2$ con el *dataset* de Mallorca, interpolado a 100 puntos por ruta usando interpolación por vecinos cercanos y reducido a 50 componentes principales mediante PCA

figuras, esta formado por las rutas más cortas, es decir, las rutas que en su zona generan un elevado número de puntos y por tanto, están mejor definidas después del proceso de interpolación. Y un segundo *cluster* (6.8b) formado por las rutas más largas (la mayoría son la vuelta a la isla) y están menos definidas puesto que la interpolación, al ser rutas más largas, separa los puntos que la forman notablemente.

¹⁰Generación de mapas: <https://github.com/bertini36/GMM/tree/master/preprocessing/maps>

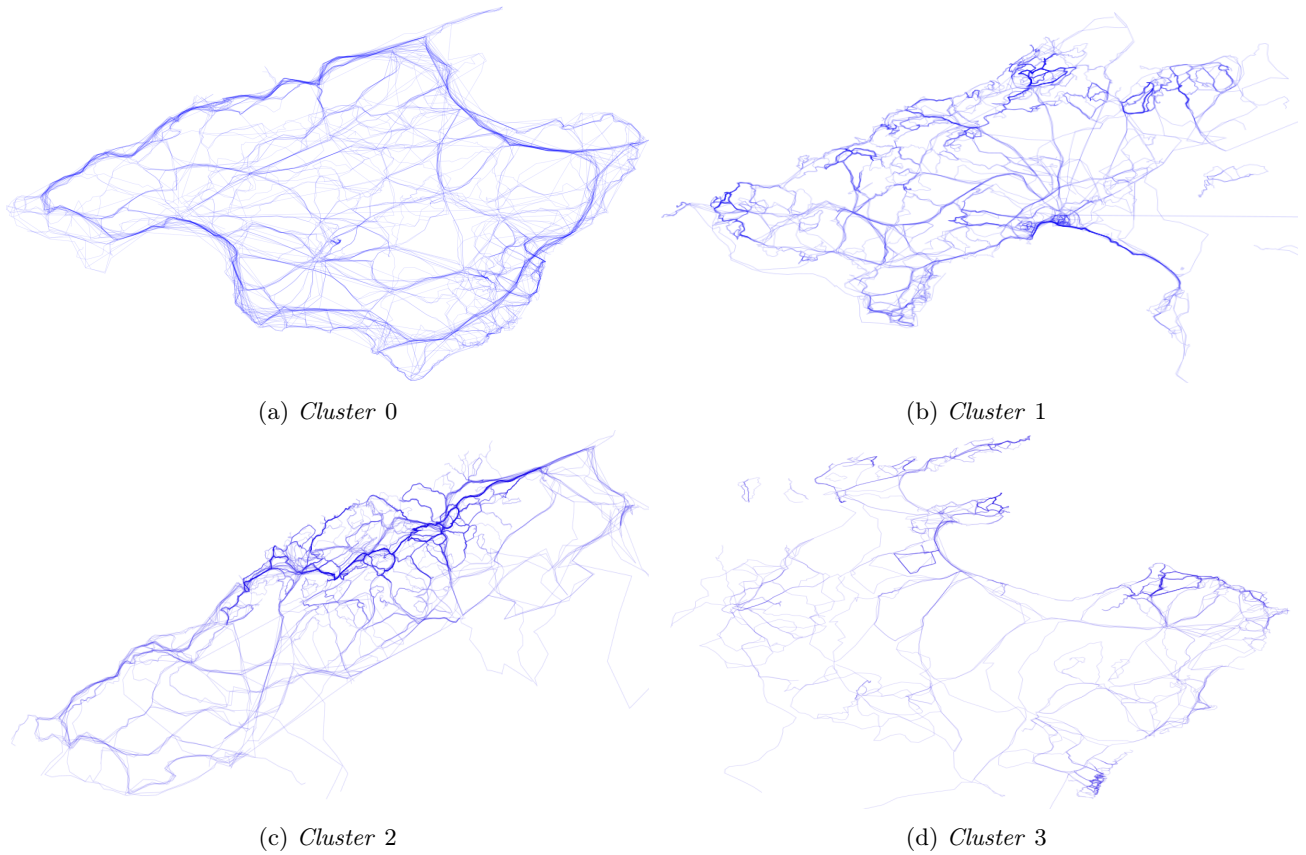
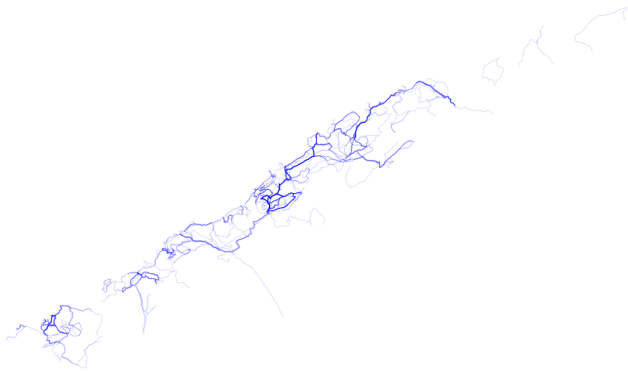
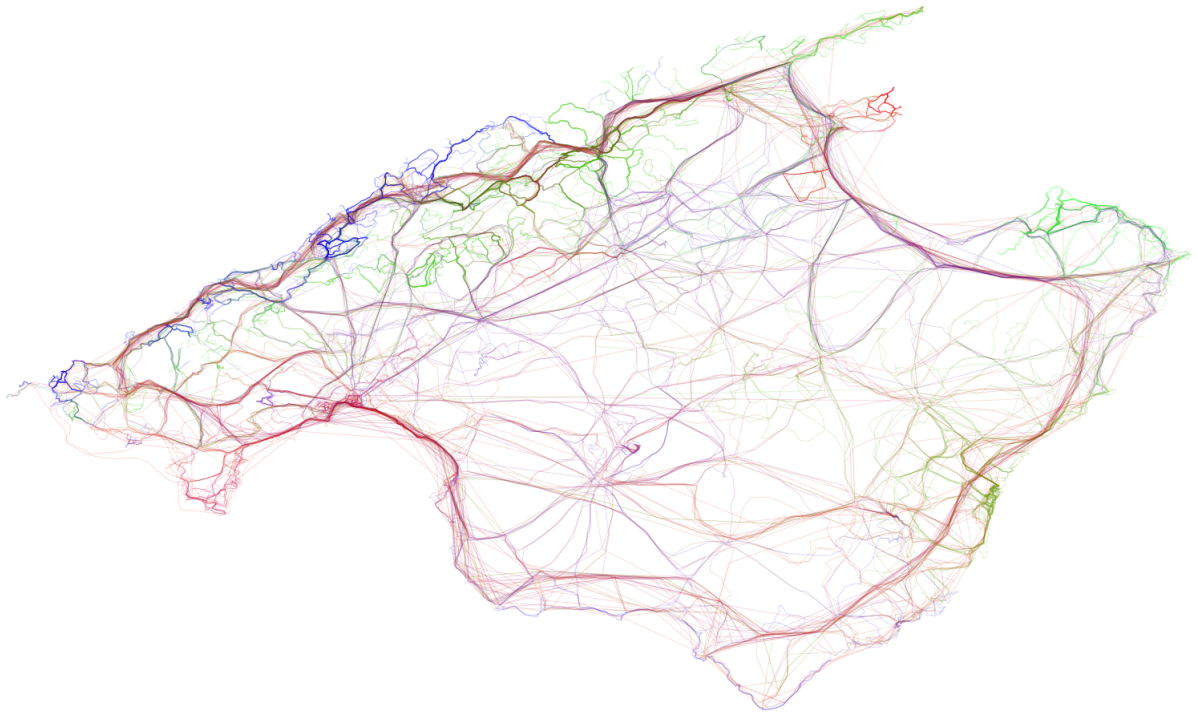


Figura 6.9: Resultados para $K = 4$ con el *dataset* de Mallorca, interpolado a 50 puntos por ruta usando interpolación por vecinos cercanos y reducido a 4 componentes principales mediante PPCA

Utilizando PPCA para $K = 4$ se descubren 3 *clusters* muy marcados por la zona geográfica (6.9b, 6.9c y 6.9d) y otro que agrupa las rutas que tienen una forma similar (6.9a).

En la figura 6.11 se pueden ver los *clusters* más relevantes de un proceso de *clustering* para $K = 30$. El algoritmo realiza las agrupaciones principalmente por zonas, pero también por similitud en la forma. El *cluster* 0 y el 26 agrupan las rutas de la *Serra de Tramuntana*, parece que el *cluster* 0 contiene las trayectorias más costeras de esa zona. El *cluster* 1 se corresponde con las rutas de la parte más al sur de la isla. El *cluster* 5 se corresponde con las rutas de la costa norte de la isla, concretamente de las largas playas de *Can Picafort* y *Son Serra de la Marina*. El *cluster* 9 se corresponde con una zona muy transitada por excursionistas, de las más bonitas de la isla para hacer senderismo, la costa entre el pueblo de *Deià* y el *Port de Soller*. El *cluster* 10 está formado por todas las rutas que dan la vuelta completa a la isla. El *cluster* 11 contiene las trayectorias que envuelven la ciudad de *Palma*. El *cluster* 21 contiene las rutas de la costa oeste de la isla.

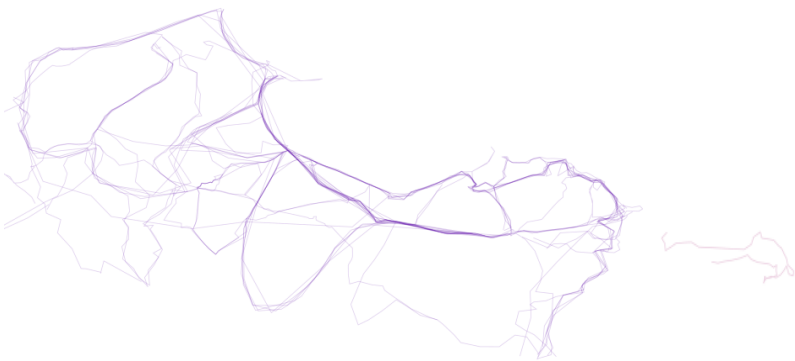
Si ahora también tenemos en cuenta la elevación de cada punto se obtienen *clusters* que agrupan rutas con una elevación media similar. En la figura 6.12 se muestran 2 agrupaciones interesantes que hace el modelo con esta nueva versión del *dataset*. Si se comparan estos dos *clusters* con la geografía de la isla de Mallorca uno se da cuenta de que el algoritmo ha identificado claramente la *Serra de Tramuntana* (*cluster* 5) gracias a la componente de elevación de los nuevos puntos del *dataset*. Además, el *cluster* 6 se corresponde con rutas a pie de costa, es decir, con una elevación muy baja, y vemos que no incluye ninguna ruta que pase por la *Serra de Tramuntana* ni por la *Serra de Llevant*.



Cluster 0



Cluster 1



Cluster 5



Cluster 9

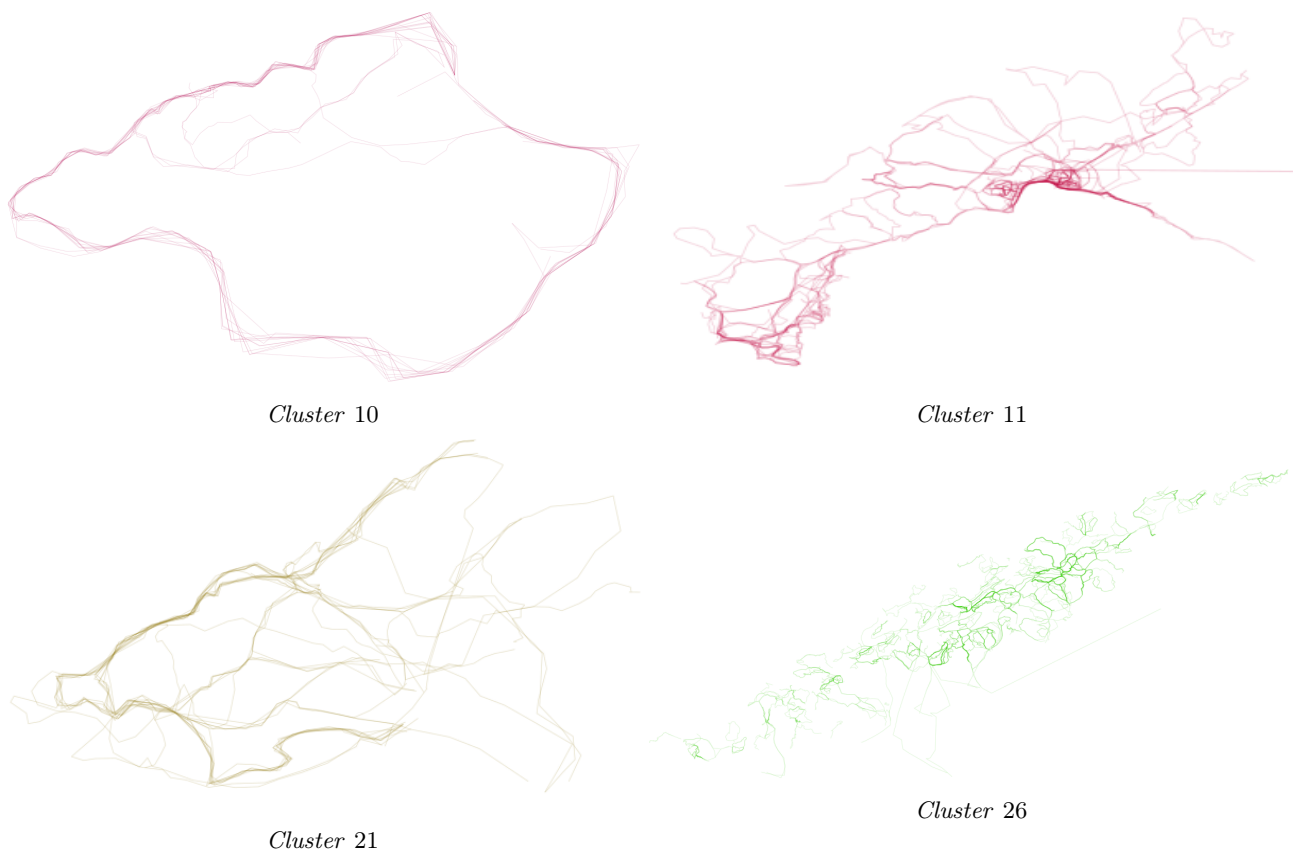


Figura 6.11: Resultados para $K = 30$ con el *dataset* de Mallorca, interpolado a 100 puntos por ruta usando interpolación por vecinos cercanos y reducido a 4 componentes principales mediante PCA

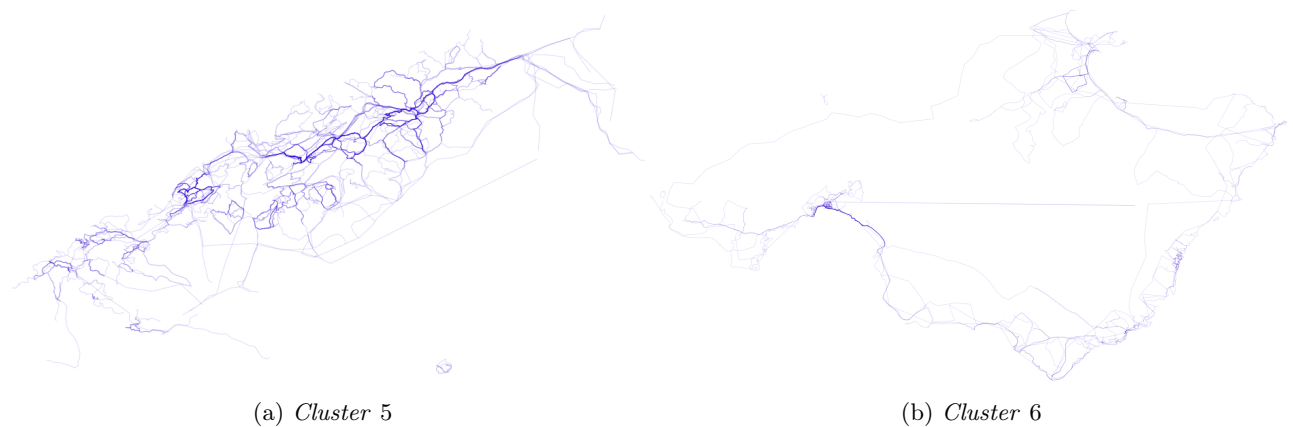


Figura 6.12: Resultados para $K = 30$ con el *dataset* de Mallorca y la información de elevación de cada punto, interpolado a 100 puntos por ruta usando interpolación por vecinos cercanos y reducido a 3 componentes mediante un *autoencoder*

6.3.3 Resultados con el dataset de Porto

Los resultados que se obtuvieron con el *dataset* de Porto no difieren en estética de los obtenidos con el *dataset* de Mallorca. Dado que el *dataset* es mucho más grande el tiempo de ejecución era mucho más elevado y por esa razón se requirió un servidor dedicado y el módulo *screen* de *Linux* para monitorizar la inferencia. En el caso de $K = 30$ tardó aproximadamente 1 semana en obtener los resultados. En el caso de $K = 2$ (figura 6.13) se aprecia que el algoritmo clasifica las rutas en función de la longitud mientras que en el caso de $K = 30$ (figura 6.15) el algoritmo también agrupa las rutas en función de su posición geográfica.

Si se comparan los resultados obtenidos en la figura 6.15 con los resultados obtenidos en el *paper* [32] (figura 14), no se puede verificar si las agrupaciones obtenidas por nuestra implementación son exactamente las mismas pero si que dichas agrupaciones, al igual que las suyas, se hacen por proximidad geográfica.

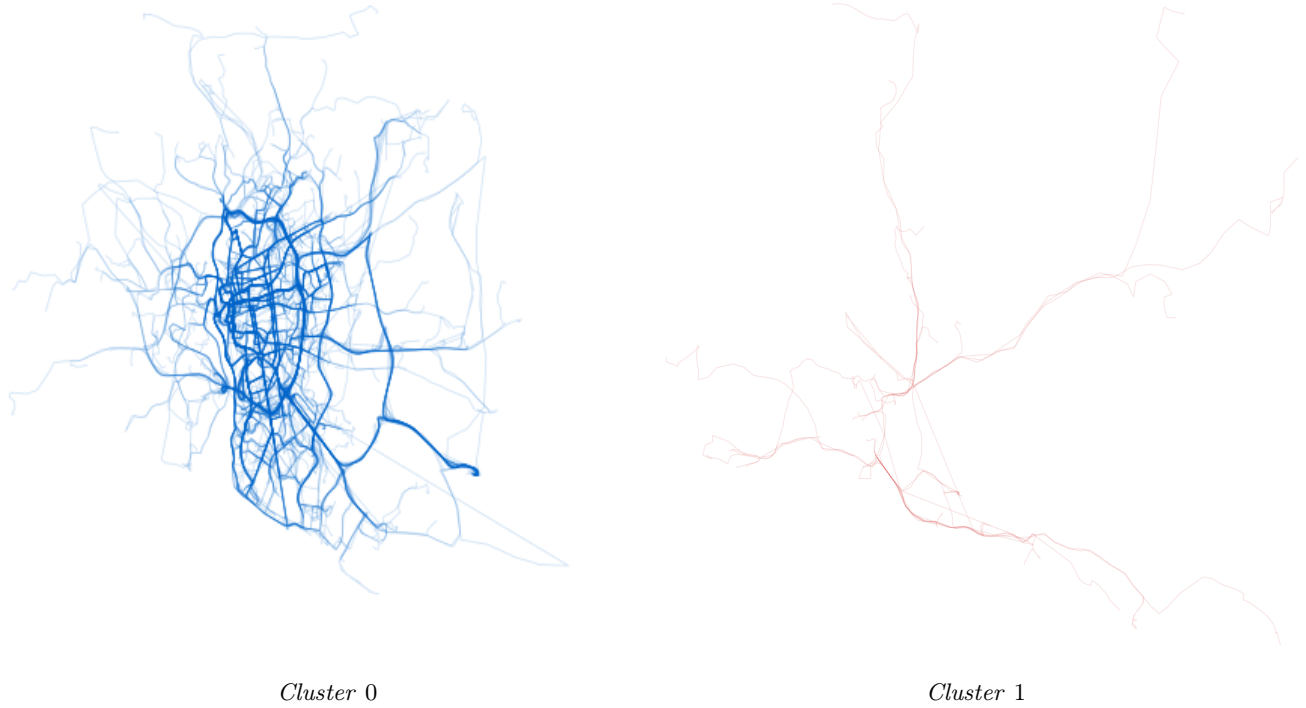


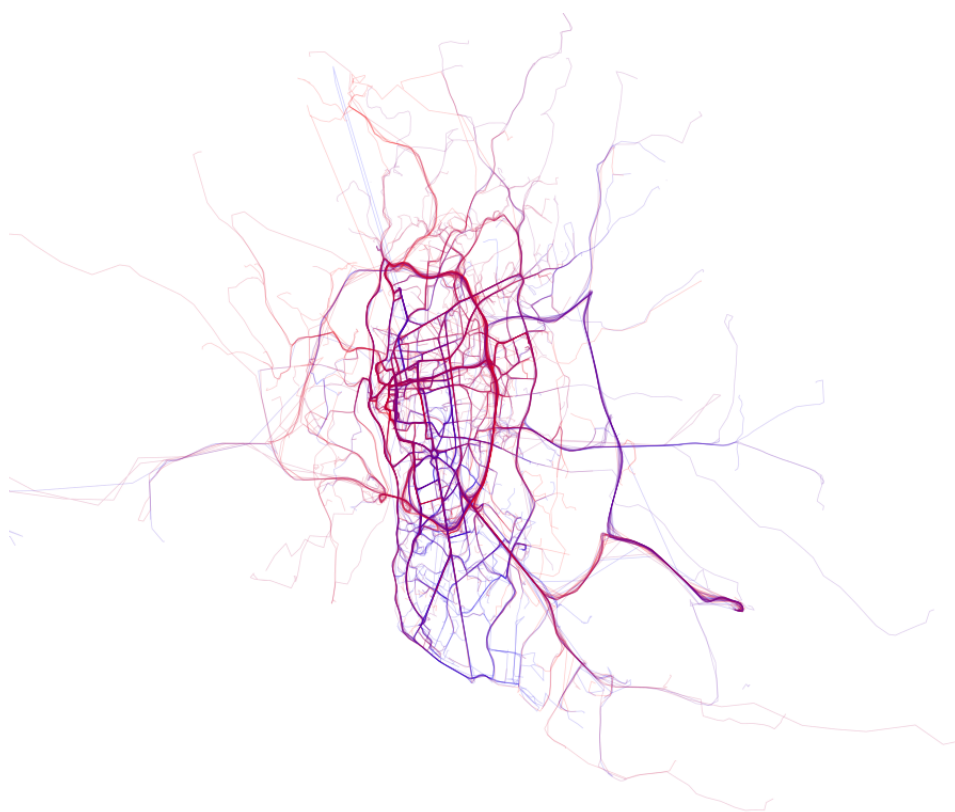
Figura 6.13: Resultados para $K = 2$ con el *dataset* de Porto, interpolado a 50 puntos por ruta usando interpolación por vecinos cercanos y reducido a 3 componentes principales mediante PCA

6.4 Generación de nuevos datos

En este apartado se comparan datos reales con datos generados mediante el proceso generativo del modelo GMM y sus distribuciones inferidas. En primer lugar se mostrarán los datos generados tras la inferencia con *datasets* sintéticos. No obstante, cuando pasamos a espacios de mayor dimensionalidad, los resultados no son tan perfectos. Para el *dataset* de Mallorca, dado que antes de la inferencia se realiza una reducción de dimensionalidad, previo a generar nuevos datos se debe realizar el procedimiento opuesto, en otras palabras, volver a aumentar la dimensionalidad de los datos. Para este proceso basta con almacenar el modelo de PCA o, en el caso del *autoencoder* guardar el *decoder*, después de realizar la reducción de dimensionalidad para utilizarlo posteriormente para el aumento de dimensionalidad. En la figura 6.17 se muestra una representación gráfica del proceso seguido para esta generación de datos.

En la figura 6.16 se muestran los resultados de generar datos mediante las distribuciones inferidas de los *datasets* sintéticos mostrados anteriormente en las figuras 6.6a y 6.6b.

En las figuras 6.18, 6.19 y 6.20 se expone una comparativa entre los datos reales del *dataset* de Mallorca y los datos generados por el algoritmo generativo. La primera pareja de figuras (6.18) se corresponde con un nuevo *dataset* de 1000 rutas generadas respetando la proporción de las agrupaciones modelada por la distribución *Dirichlet*. Las otras dos comparativas (6.19 y 6.20) son de rutas generadas para unos *clusters* concretos.



Cluster 0



Cluster 3



Cluster 4



Cluster 11



Figura 6.15: Resultados para $K = 30$ con el *dataset* de Porto, interpolado a 50 puntos por ruta usando interpolación por vecinos cercanos y reducido a 3 componentes principales mediante PCA

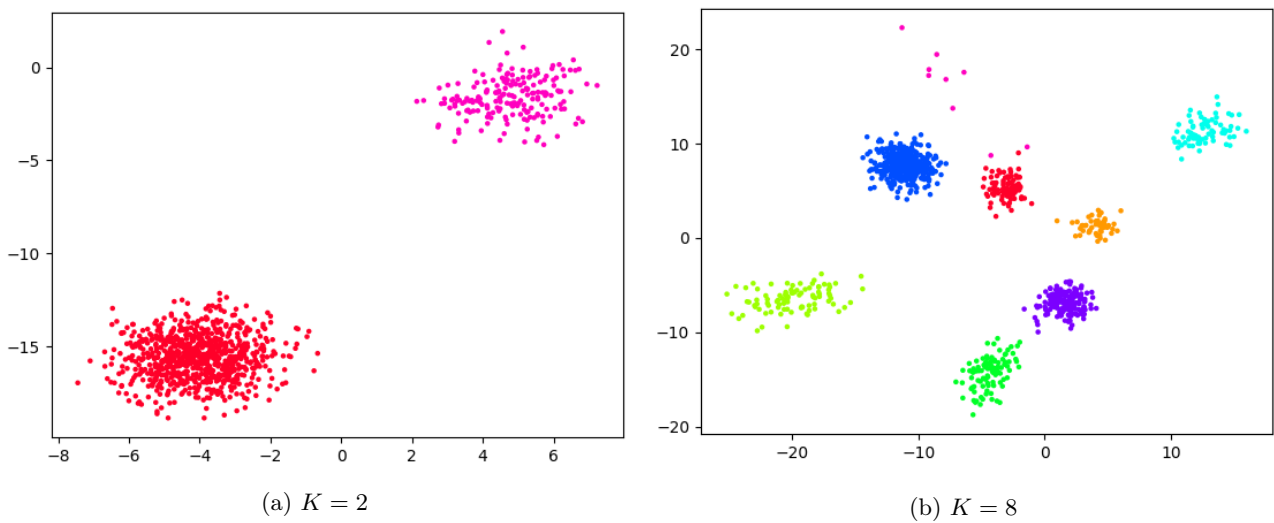


Figura 6.16: Generación de nuevos datos para los *datasets* sintéticos mediante las distribuciones inferidas por el modelo GMM

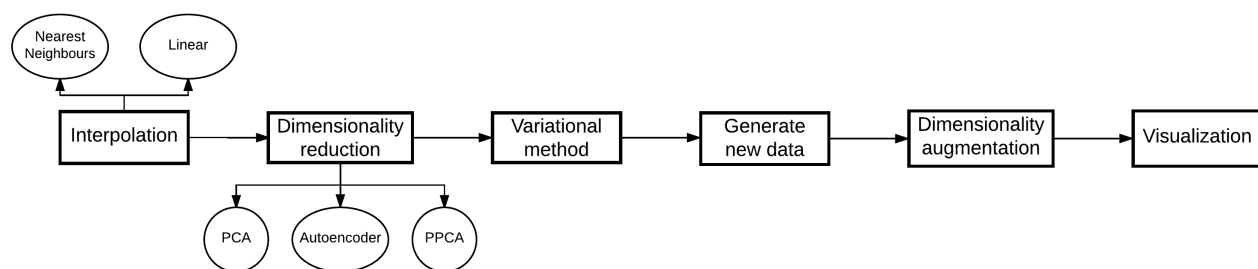


Figura 6.17: Proceso seguido para la generación de nuevos datos

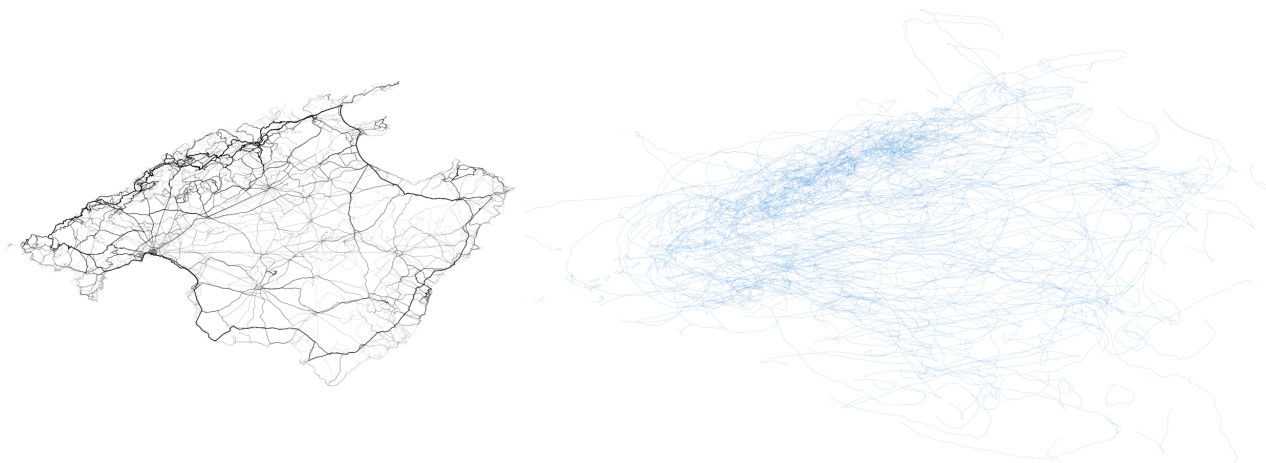


Figura 6.18: Comparativa de datos reales con datos generados



Figura 6.19: Comparativa de datos reales con datos generados del *cluster* 9

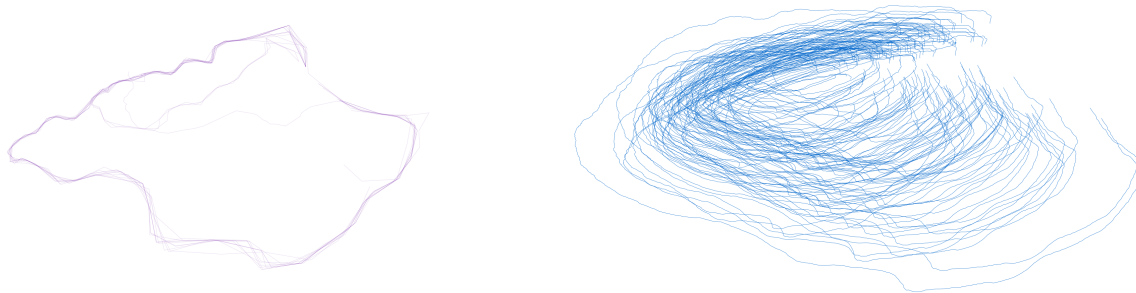


Figura 6.20: Comparativa de datos reales con datos generados del *cluster* 11

7. Conclusiones

En este capítulo se comentan las conclusiones a las que se han llegado durante el desarrollo del proyecto y las líneas abiertas que existen actualmente en estos temas.

7.1 Lecciones aprendidas

Una de las principales motivaciones para este proyecto era introducir conocimiento sobre este tipo de modelos en el *Barcelona Supercomputing Center* (BSC). Todo empezó con el descubrimiento de la librería *Edward*¹, una librería para *probabilistic programming* desarrollada por un estudiante de doctorado de la universidad de Columbia llamado Dustin Tran². La implementación de esta librería esta bajo la supervisión del departamento de David Blei³. Los miembros de este departamento y cada vez más gente de la comunidad empiezan a utilizar esta librería para el desarrollo e inferencia de modelos probabilísticos principalmente porque, al estar implementada sobre *Tensorflow*, permite entrenar los modelos sobre *Graphic Processing Unit* (GPU), *hardware* que actualmente está ofreciendo un mejor rendimiento para este tipo de tareas. Dado quiénes estaban detrás de su desarrollo y la aceptación que estaba recibiendo nos pareció buena idea adoptar el uso de esta librería en el departamento para la implementación de este tipo de modelos. En el *Autonomic Systems and e-Business Platforms group* del BSC, *Tensorflow* es una de las librerías más utilizadas y por tanto, *Edward*, al estar implementada en *Tensorflow*, nos permitía acercarnos al resto del equipo y unir el mundo del *probabilistic machine learning* con el del *deep learning*.

El estudio de la evolución de los métodos variacionales, para la inferencia de modelos probabilísticos, nos ha permitido ver que, dadas todas las suposiciones que hace, *Black Box Variational Inference* (BBVI) es un método sometido a una elevada varianza y a unas convergencias lentas por culpa, principalmente, de la aproximación de las integrales, mediante *Monte-Carlo integration*, de la *Evidence Lower Bound* (ELBO). Posiblemente esto es lo que provoca que un modelo como el *Gaussian Mixture Model* (GMM), implementado con la librería *Edward*, no de el resultado esperado cuando se pasa a espacios más complejos, se aumenta el tamaño del *dataset*, o se buscan un número elevado de *mixturas*.

Se ha podido corroborar que si eres capaz de derivar el modelo siempre vas a obtener unos resultados mejores y más rápidamente. No obstante, la derivación de un modelo probabilístico no es una tarea trivial. La derivación del modelo GMM, el cual no es un modelo demasiado complejo, ya supone disponer de unos conocimientos matemáticos, concretamente estadísticos, bastante sólidos.

La realización de las pruebas, cuyos resultados han sido mostrados en el apartado de experimentación, entre algoritmos que actualizan los parámetros del modelo de forma analítica y los que lo hacen mediante la optimización por gradientes; o entre algoritmos estocásticos y no estocásticos nos ha permitido obtener una serie de conclusiones que se comentan a continuación:

- Si existe la posibilidad de derivar las actualizaciones de los parámetros variacionales siempre se van a obtener unas convergencias mejores, en un menor número de iteraciones y con un consumo de memoria inferior.
- En el caso de no poder realizar las derivaciones una de las alternativas existentes pasa por utilizar *Automatic Differentiation* (AD) con herramientas como *Tensorflow*, *Autograd* o *Theano*.
- Si la función a derivar, en el caso de *Variational Inference* (VI) la ELBO, depende directamente de los datos y el *dataset* es muy grande; el uso de *Tensorflow* no es la mejor opción. En estos casos conviene

¹ *Edward*: <https://github.com/blei-lab/edward>

² Dustin Tran: <http://dustintran.com/>

³ David Blei: <http://www.cs.columbia.edu/~blei/>

adaptar los algoritmos a su versión estocástica.

- En el caso del modelo probabilístico GMM, el optimizador por gradientes que obtiene las mejores convergencias es *RMSprop*.
- Si se dispone de una tarjeta gráfica de *Nvidia* con soporte de *Compute Unified Device Architecture* (CUDA), *Tensorflow* pasa a ser una alternativa muy conveniente para la implementación del algoritmo. No obstante, si la función a derivar es muy compleja y utiliza funciones poco populares de esta librería no se obtendrá la mejora esperada. Con el tiempo este problema desaparecerá ya que se espera que *Tensorflow* disponga de implementaciones para CUDA de todas sus funciones.

7.2 ¿Qué he aprendido?

Este proyecto me ha permitido meterme de lleno en el mundo del *machine learning*, motivo principal por el cuál quería acceder a una institución como el BSC. Trabajar en el BSC me ha permitido trabajar con tecnologías muy punteras como *Tensorflow*, *Edward* o CUDA, las cuales, al llegar a Barcelona ni siquiera conocía. Saber utilizar estas tecnologías, entender los diferentes paradigmas que utilizan y conocer exactamente cuál es su función y cuándo vale la pena utilizarlas creo que es un conocimiento muy importante que he adquirido y que posiblemente me sirva en un futuro próximo.

Otro problema que ha ido despertando mi interés y que desconocía completamente al empezar el proyecto ha sido el de la aproximación de la distribución *posterior*. Las diferentes aproximaciones algorítmicas que existen actualmente para solventar este problema (*Markov Chain Monte Carlo* (MCMC) y VI) junto con los modelos generativos me parecen muy interesantes para la investigación y con una aplicabilidad muy interesante.

El uso de *Tensorflow* para el desarrollo fue un gran acierto. El interés que me despertó esta librería me permitió meterme de lleno en el campo del *deep learning*. Esto implicó entender como funciona una red neuronal, los diferentes tipos que existen, para qué utilizar cada una y sobretodo saber implementarlas usando esta librería. Gracias a estos conocimientos no nos fue difícil implementar un *autoencoder* para la etapa de reducción de dimensionalidad de datos reales.

Pero sobretodo, de lo que más orgulloso me siento, es de haber aprendido sobre modelos probabilísticos. Quiero recalcar que mi trabajo ha consistido en entender las derivaciones para poder programarlas, explicarlas y documentarlas paso a paso para esta documentación con el objetivo de que sirviera de punto de partida para futuras investigaciones del BSC.

Todo esto me ha permitido leer y entender *papers* muy interesantes, los cuales, antes de este proyecto no hubiera entendido ni el título. También me permitió implementar algunos modelos probabilísticos con *Edward*⁴⁵ que me permitieron convertirme en contribuyente de esta librería. Finalmente, el enfrentarme a todas las ecuaciones mostradas me ha permitido recordar y mejorar enormemente mi nivel en estadística y matemáticas.

7.3 Dificultades

Como ya se ha podido ver en diversas ocasiones en esta documentación, el *machine learning*, más concretamente el *probabilistic machine learning*, incumbe un alto nivel de matemáticas. Esta barrera de entrada matemática y estadística me supuso el obstáculo más grande para el desarrollo de este proyecto. Entender las propiedades de la *Exponential Family*, las derivaciones del modelo GMM, las diferentes distribuciones implicadas, el concepto de VI, ... Todos ellos supusieron un gran tiempo de aprendizaje y asimilación antes de empezar a implementar cualquier cosa.

Otro problema que se tuvo que sobrellevar fue el uso de tecnologías muy recientes como *Edward* y *Tensorflow* (apenas tienen un año cada una de ellas). Como con cualquier *software* joven hubo que lidiar con *bugs* que se solucionaron al cabo de un tiempo o que todavía están en la lista de *issues* de la librería. Hay una frase que define muy bien esto: "El *hardware* es aquello que con el tiempo deja de funcionar mientras que el *software* es aquello que con el tiempo funciona". No me cabe duda de que estas dos librerías mejorarán enormemente en los próximos años y sobretodo mejorarán gracias a las comunidades que se están formando a su alrededor y que

⁴*Dirichlet-Categorical model*: https://github.com/blei-lab/edward/blob/master/examples/dirichlet_categorical.py

⁵*Invgamma-Normal model*: https://github.com/blei-lab/edward/blob/master/examples/invgamma_normal_mh.py

cada día son mas grandes.

Por un lado, muchos de los problemas que nos encontrábamos en la librería *Edward* se debían al bajo nivel de convergencia del algoritmo de BBVI, a la aparición de infinitos durante la inferencia, al cambio de sintaxis de una versión a otra, a las limitaciones para expresar ciertas restricciones numéricas o la no disponibilidad de algunas distribuciones estadísticas para la modelización. Muchos de estos problemas se notificaron por *Github* a la librería y de muchos ellos recibimos *feedback* por parte de Dustin Tran.

Por otro lado, el principal problema que tuve con *Tensorflow* fue su paradigma, el cual es bastante diferente a todo lo que había programado hasta la fecha. El hecho de primero especificar un grafo de operaciones, para luego especificar qué partes del grafo ejecutar supuso ser una metodología completamente nueva para mí.

Finalmente, trabajar por primera vez en un ámbito completamente de investigación también supuso un gran reto. Buscar respuestas en *Google* a un cierto problema y no encontrar prácticamente nada es bastante frustrante. Aún así, por la satisfacción que supone conseguir encontrar un error o de hacer algo que todavía nadie había probado, vale la pena.

7.4 Futuros pasos

Creo que la investigación en estos temas debería ir dirigida a combinar el *deep learning* con el *probabilistic machine learning*. Estar en el BSC me permitió poder asistir, el pasado mes de diciembre, al *Neural Information Processing Systems* (NIPS) que se celebró en Barcelona. En esta conferencia se presentan los avances anuales en la mayoría de campos del *machine learning* y una de las cosas que estaba al orden del día era la composición de modelos que combinaban estos dos mundos. Los *variational autoencoders* [55] son un buen ejemplo de ello.

Otro ejemplo es el *paper* de Matthew James Johnson de la universidad de *Harvard* [56]. En este *paper* explican como utilizar redes neuronales para mejorar el modelo tradicional de GMM. En las figura 7.1 se puede apreciar la diferencia en el resultado de un *clustering* con estos modelos. El problema de GMM es que no representa el *clustering* natural de los datos, es inflexible. La versión que ellos proponen consiste en transformar los datos mediante una red neuronal, es decir, utilizar un modelo GMM formado por observaciones no lineales. Esta combinación de la flexibilidad que ofrece el *deep learning* con los modelos probabilísticos permite agrupar los datos de la forma que vemos en la figura 7.1b.

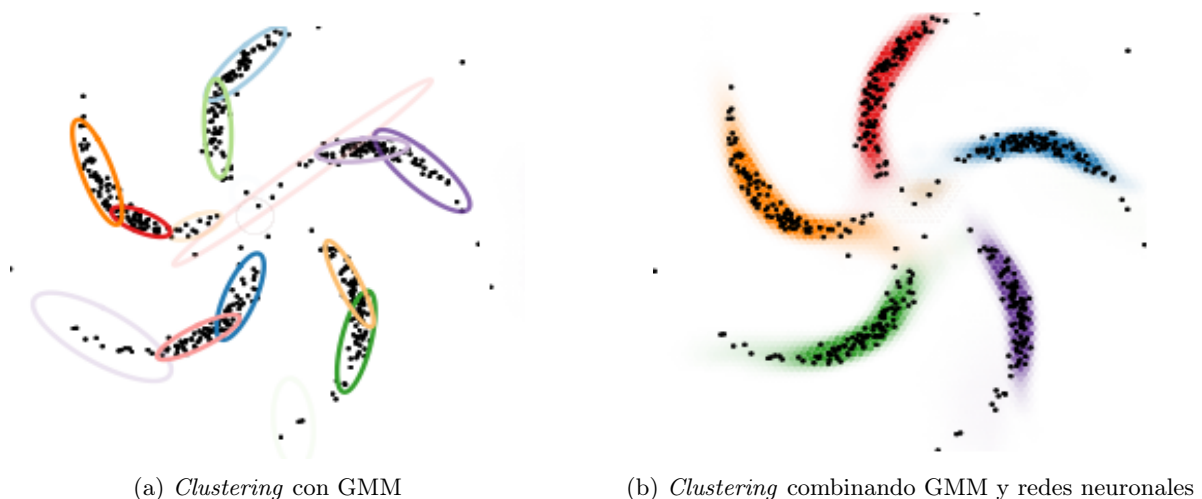


Figura 7.1: (Fuente: [56])

También creo que apostar por librerías como *Edward* y en extensión *Tensorflow*, ahora que parece que se esta adaptando al resto de campos del *machine learning*, es una buena decisión. Actualmente, uno de los problemas a los que se enfrenta cualquier algoritmo de *machine learning* es la dimensionalidad del *dataset*. Todo lo que se construya sobre una librería como *Tensorflow*, que permite la distribución y utiliza GPUs para paralelizar sus cálculos, es una gran aportación en este campo.

A. Apéndices

A.1 Apéndices del capítulo 2

A.1.1 Derivación completa de la evidencia para Variational Inference (VI)

$$\begin{aligned}
p(\theta|x) &= \frac{p(x|\theta)p(\theta)}{p(x)} \Rightarrow p(x) = \frac{p(x, \theta)}{p(\theta|x)} \Rightarrow \\
\ln p(x) &= \ln \frac{p(x, \theta)}{p(\theta|x)} \Rightarrow \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{p(\theta|x)} d\theta \Rightarrow \\
\int q(\theta|\lambda) \ln \frac{p(x, \theta)}{p(\theta|x)} \frac{q(\theta|\lambda)}{q(\theta|\lambda)} d\theta &\Rightarrow \int q(\theta|\lambda) \left(\ln \frac{q(\theta|\lambda)}{p(\theta|x)} + \ln \frac{p(x, \theta)}{q(\theta|\lambda)} \right) d\theta \Rightarrow \\
\ln p(x) &= \int q(\theta|\lambda) \ln \frac{q(\theta|\lambda)}{p(\theta|x)} d\theta + \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta
\end{aligned}$$

A.1.2 Reescritura de la ELBO

$$\begin{aligned}
ELBO(q(\theta|\lambda), p(x, \theta)) &= \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta \\
&= \int q(\theta|\lambda) \ln p(x, \theta) d\theta - \int q(\theta|\lambda) \ln q(\theta|\lambda) d\theta \\
&= \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))
\end{aligned}$$

A.1.3 Derivación genérica completa de un parámetro variacional

$$\begin{aligned}
q(\pi|\lambda_\pi) &\propto \exp(\mathbb{E}_{q(c|\lambda_c)q(\mu|\lambda_\mu)}(\ln(p(x, c, \mu|\alpha, m_0, \beta_0, \Delta)))) \\
&\propto \exp\left(\ln(p(\pi|\alpha)) + \sum_{n=1}^N \mathbb{E}_{q(c_n|\lambda_{c_n})}(\ln(p(c_n|\pi)))\right) \\
&\propto \exp\left(\sum_{k=1}^K (\alpha_k - 1) \ln(\pi_k) + \sum_{n=1}^N \sum_{k=1}^K \mathbb{E}_{q(c_n|\lambda_{c_n})}(\mathbb{I}(c_n = k)) \ln(\pi_k)\right) \\
&\propto \exp\left(\sum_{k=1}^K (\alpha_k - 1) + \sum_{n=1}^N \mathbb{E}_{q(c_n|\lambda_{c_n})}(\mathbb{I}(c_n = k)) \ln(\pi_k)\right)
\end{aligned}$$

A.1.4 Aproximación de una distribución variacional suponiendo mean-field assumption

En primer lugar se reescribe la *Evidence Lower Bound* (ELBO) como:

$$\begin{aligned}
ELBO(q(\theta|\lambda), p(x, \theta)) &= \int q(\theta|\lambda) \ln \frac{p(x, \theta)}{q(\theta|\lambda)} d\theta \\
&= \int \prod_{i=1}^N q(\theta_i|\lambda_i) \left(\ln p(x, \theta) - \sum_{i=1}^N \ln q(\theta_i|\lambda_i) \right) d\theta \\
&= \int q(\theta_j|\lambda_j) \prod_{i=0, i \neq j}^N q(\theta_i|\lambda_i) (\ln p(x, \theta) - \ln q(\theta_j|\lambda_j)) d\theta \\
&\quad - \int q(\theta_j|\lambda_j) \prod_{i=0, i \neq j}^N q(\theta_i|\lambda_i) \sum_{i=0, i \neq j}^N \ln q(\theta_i|\lambda_i) d\theta \\
&= \int q(\theta_j|\lambda_j) \left(\int \prod_{i=0, i \neq j}^N q(\theta_i|\lambda_i) \ln p(x|\theta) d\theta_{\neq j} - \ln q(\theta_j|\lambda_j) \right) d\theta_j \\
&\quad - \int q(\theta_j|\lambda_j) \int \prod_{i=0, i \neq j}^N q(\theta_i|\lambda_i) \ln \prod_{i=0, i \neq j}^N q(\theta_i|\lambda_i) d\theta_{\neq j} d\theta_j \\
&= \int q(\theta_j|\lambda_j) \ln \frac{\exp(\mathbb{E}_{q(\theta_{\neq j}|\lambda_{\neq j})}(\ln p(x|\theta)))}{q(\theta_j|\lambda_j)} d\theta_j + \mathbb{C} \\
&= -KL[q(\theta_j|\lambda_j) || \exp(\mathbb{E}_{q(\theta_{\neq j}|\lambda_{\neq j})}(\ln p(x|\theta)))] + \mathbb{C}
\end{aligned}$$

Si a continuación suponemos las $q(\theta_{\neq j}|\lambda_{\neq j})$ fijas, la $q(\theta_j|\lambda_j)$ que maximiza la ELBO queda como:

$$\begin{aligned}
q(\theta_j|\lambda_j) &= \underset{q(\theta_j|\lambda_j)}{\operatorname{argmax}} [ELBO(q(\theta|\lambda), p(x, \theta))] \\
&= \frac{1}{Z} \exp(\mathbb{E}_{q(\theta_{\neq j}|\lambda_{\neq j})}(\ln p(x, \theta)))
\end{aligned}$$

Z representa un sesgo, por tanto, si lo obviemos, la aproximación de $q(\theta_j|\lambda_j)$ queda como:

$$q(\theta_j|\lambda_j) \propto \exp(\mathbb{E}_{q(\theta_{\neq j}|\lambda_{\neq j})}(\ln p(x, \theta)))$$

A.1.5 Formulación de una distribución Dirichlet en Exponential Family

$$\begin{aligned}
p(\pi|\alpha) &= \exp\left(\ln\left(\frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)}\right) \prod_{k=1}^K \pi_k^{\alpha_k-1}\right) \\
&= \exp\left(\ln\left(\Gamma\left(\sum_{k=1}^K \alpha_k\right)\right) - \ln\left(\prod_{k=1}^K \Gamma(\alpha_k)\right) + \ln\left(\prod_{k=1}^K \pi_k^{\alpha_k-1}\right)\right) \\
&= \exp\left(\ln\left(\Gamma\left(\sum_{k=1}^K \alpha_k\right)\right) - \ln\left(\prod_{k=1}^K \Gamma(\alpha_k)\right) + \sum_{k=1}^K (\alpha_k - 1) \pi_k\right) \\
&= \exp\left(\ln\left(\Gamma\left(\sum_{k=1}^K \alpha_k\right)\right) - \ln\left(\prod_{k=1}^K \Gamma(\alpha_k)\right) + \begin{bmatrix} \alpha_1 - 1 \\ \vdots \\ \alpha_K - 1 \end{bmatrix}^T \begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_K) \end{bmatrix}\right)
\end{aligned}$$

A.1.6 Derivación score gradients de la ELBO

$$\begin{aligned}
\nabla_{\lambda} ELBO(q(\theta|\lambda), p(x, \theta)) &= \nabla_{\lambda} \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda)) \\
&= \nabla_{\lambda} \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] q(\theta|\lambda) d\theta \\
&= \int \nabla_{\lambda} [\ln p(x, \theta) - \ln q(\theta|\lambda)] q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \int [\nabla_{\lambda} \ln p(x, \theta) - \nabla_{\lambda} \ln q(\theta|\lambda)] q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \int [0 - \nabla_{\lambda} \ln q(\theta|\lambda)] q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \int -\frac{\nabla_{\lambda} q(\theta|\lambda)}{q(\theta|\lambda)} q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \int -\nabla_{\lambda} q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \nabla_{\lambda} \int -q(\theta|\lambda) d\theta + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \nabla_{\lambda} (-1) + \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta \\
&= \int [\ln p(x, \theta) - \ln q(\theta|\lambda)] \nabla_{\lambda} q(\theta|\lambda) d\theta
\end{aligned}$$

A.2 Apéndices del capítulo 4

A.2.1 Distribuciones del modelo GMM formuladas en términos de la Exponential Family

Distribución	Base measure	Natural Params.	Suff. Statistics	Cumulant
<i>Dirichlet</i> $p(\pi \alpha)$	1	$\begin{bmatrix} \alpha_1 - 1 \\ \vdots \\ \alpha_k - 1 \end{bmatrix}$	$\begin{bmatrix} \ln \pi_1 \\ \vdots \\ \ln \pi_k \end{bmatrix}$	$\sum_{i=1}^k \log \Gamma(\alpha_i) - \log \Gamma\left(\sum_{j=1}^K (\alpha_j)\right)$
<i>Dirichlet</i> $p(\pi \eta)$	1	$\begin{bmatrix} \eta_1 + 1 \\ \vdots \\ \eta_k + 1 \end{bmatrix}$	$\begin{bmatrix} \ln \pi_1 \\ \vdots \\ \ln \pi_k \end{bmatrix}$	$\sum_{i=1}^k \log \Gamma(\eta_i + 1) - \log \Gamma\left(\sum_{j=1}^K (\eta_j + 1)\right)$
<i>Categorical</i> $p(c \pi)$	1	$\begin{bmatrix} \ln \pi_1 \\ \vdots \\ \ln \pi_k \end{bmatrix}$	$\begin{bmatrix} \mathbb{I}(c = 1) \\ \vdots \\ \mathbb{I}(c = k) \end{bmatrix}$	0
<i>Categorical</i> $p(c \eta)$	1	$\begin{bmatrix} e^{\eta_1} \\ \vdots \\ e^{\eta_k} \end{bmatrix}$	$\begin{bmatrix} \mathbb{I}(c = 1) \\ \vdots \\ \mathbb{I}(c = k) \end{bmatrix}$	0
<i>Normal</i> $p(x \mu, \Sigma)$	$(2\pi)^{-\frac{D}{2}}$	$\begin{bmatrix} \Sigma^{-1}\mu \\ -\frac{1}{2}\Sigma^{-1} \end{bmatrix}$	$\begin{bmatrix} x \\ xx^T \end{bmatrix}$	$\frac{1}{2}\mu^T \Sigma^{-1} \mu + \frac{1}{2} \ln \Sigma $
<i>Normal</i> $p(x \eta_1, \eta_2)$	$(2\pi)^{-\frac{D}{2}}$	$\begin{bmatrix} -\frac{1}{2}\eta_2^{-1}\eta_1 \\ -\frac{1}{2}\eta_2^{-1} \end{bmatrix}$	$\begin{bmatrix} x \\ xx^T \end{bmatrix}$	$-\frac{1}{4}\eta_1^T \eta_2^{-1} \eta_1 - \frac{1}{2} \ln -2\eta_2 $
<i>Normal Inverse Wishart</i> $p(\mu, \Sigma m_0, \beta_0, \nu_0, W_0)$	$(2\pi)^{-\frac{D}{2}}$	$\begin{bmatrix} m_0^T \beta_0 \\ W_0 + \beta_0 m_0 m_0^T \\ \beta_0 \\ \nu_0 + D + 2 \end{bmatrix}$	$\begin{bmatrix} \Sigma^{-1}\mu \\ -\frac{1}{2}\Sigma^{-1} \\ -\frac{1}{2}\mu^T \Sigma^{-1} \mu \\ -\frac{1}{2} \ln \Sigma \end{bmatrix}$	$\frac{\nu_0 D}{2} \ln 2 + \ln \Gamma_D\left(\frac{\nu_0}{2}\right) - \frac{D}{2} \log \beta_0 - \frac{\nu_0}{2} \log W_0 $
<i>Normal Inverse Wishart</i> $p(\mu, \Sigma \eta_1, \eta_2, \eta_3, \eta_4)$	$(2\pi)^{-\frac{D}{2}}$	$\begin{bmatrix} \frac{\eta_1}{\eta_3} \\ \eta_2 - \frac{\eta_1}{\eta_3} \eta_1 \eta_1^T \\ \eta_3 \\ \eta_4 - D - 2 \end{bmatrix}$	$\begin{bmatrix} \Sigma^{-1}\mu \\ -\frac{1}{2}\Sigma^{-1} \\ -\frac{1}{2}\mu^T \Sigma^{-1} \mu \\ -\frac{1}{2} \ln \Sigma \end{bmatrix}$	$\frac{(\eta_4 - D - 2)^D}{2} \ln 2 + \ln \Gamma_D\left(\frac{\eta_4 - D - 2}{2}\right) - \frac{1}{2} \ln \eta_3 - \frac{\eta_4 - D - 2}{2} \ln \left \eta_2 - \frac{1}{\eta_3 \eta_1 \eta_1^T} \right $

A.2.2 Cálculo de expectations

En este apéndice se resuelven una serie de *expectations* necesarias para la derivación de las actualizaciones de los parámetros variacionales y para la derivación de la ELBO. Gracias a las propiedades de la *Exponential Family*, la *expectation* de una distribución de esta familia se puede calcular como la *expectation* de sus *natural parameters*.

$$\mathbb{E}[p(x)] = \mathbb{E}_{p(x)} t(x)$$

Además, esta *expectation* de los *natural parameters* se puede calcular como la derivada del cumulante respecto a los *natural parameters*:

$$\mathbb{E}_{p(x)} t(x) = \frac{\partial a(\eta)}{\partial \eta}$$

Haciendo uso de esta igualdad y de las formulaciones de las distribuciones en términos de η mostradas en el apéndice A.2.1 se derivará el cumulante de cada una de las distribuciones que conforman el modelo *Gaussian Mixture Model* (GMM).

Expectation de los sufficient statistics de una distribución Categorical

Para obtener esta *expectation* no hace falta utilizar la igualdad comentada anteriormente. Simplemente:

$$\mathbb{E}_{p(c|\pi)} t(c) = \mathbb{E}_{p(c|\pi)} \begin{bmatrix} \mathbb{I}(c=1) \\ \vdots \\ \mathbb{I}(c=k) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^K \mathbb{I}(c_i=1) \prod_{j=1}^K \pi_j^{\mathbb{I}(c_i=j)} \\ \vdots \\ \sum_{i=1}^K \mathbb{I}(c_i=k) \prod_{j=1}^K \pi_j^{\mathbb{I}(c_i=j)} \end{bmatrix} = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_k \end{bmatrix}$$

Expectation de los sufficient statistics de una distribución Dirichlet

En primer lugar se define la función *Digamma*, la cual será de suma importancia en el cálculo de esta *expectation*:

$$\Psi(x) = \frac{\partial \ln \Gamma(x)}{\partial x}$$

Cálculo de la *expectation* de los *sufficient statistics* parametrizados por η (véase el apéndice A.2.1):

$$\mathbb{E}_{p(\pi|\alpha)} t(\pi) = \mathbb{E}_{p(\pi|\alpha)} \begin{bmatrix} \ln \pi_1 \\ \vdots \\ \ln \pi_k \end{bmatrix} = \frac{\partial a(\eta)}{\partial \eta}$$

Entonces, para un k concreto se tiene:

$$\begin{aligned} \mathbb{E}_{p(\pi|\alpha)} \ln \pi_k &= \frac{\partial a(\eta_k)}{\partial \eta_k} \\ &= \frac{\partial \ln \Gamma(\eta_k + 1)}{\partial \eta_k} - \frac{\partial \ln \Gamma\left(\sum_{i=1}^K \eta_i + 1\right)}{\partial \eta_k} \\ &= \Psi(\eta_k + 1) - \Psi\left(\sum_{i=1}^K \eta_i + 1\right) \end{aligned}$$

Y se reescribe en función del parámetro real de la distribución *Dirichlet*:

$$\mathbb{E}_{p(\pi|\alpha)} \ln \pi_k = \Psi(\alpha_k) - \Psi\left(\sum_{i=1}^K \alpha_i\right)$$

Expectation de los sufficient statistics de una distribución Normal

Cálculo de la *expectation* de los *sufficient statistics* parametrizados por η (véase el apéndice A.2.1):

$$\mathbb{E}_{p(x|\mu, \Sigma)} t(x) = \mathbb{E}_{p(x|\mu, \Sigma)} \begin{bmatrix} x \\ xx^T \end{bmatrix} = \frac{\partial a(\eta)}{\partial \eta}$$

Expectation del sufficient statistic x

$$\begin{aligned}\mathbb{E}_{p(x|\mu,\Sigma)}[x] &= \frac{\partial a(\eta_1, \eta_2)}{\partial \eta_1} \\ &= -\frac{1}{4}2\eta_2^{-1}\eta_1\end{aligned}$$

Y se sustituyen los parámetros η_1 y η_2 por los parámetros reales de la distribución *Normal*:

$$\mathbb{E}_{p(x|\mu,\Sigma)}[x] = -\frac{1}{2}(-2\Sigma)\Sigma^{-1}\mu = \mu$$

Se trata de un resultado importante puesto que se acaba de demostrar que la *expectation* de una distribución *Normal* es su media.

Expectation del sufficient statistic xx^T .

$$\begin{aligned}\mathbb{E}_{p(x|\mu,\Sigma)}[xx^T] &= \frac{\partial a(\eta_1, \eta_2)}{\partial \eta_2} \\ &= \frac{1}{4}\eta_2^{-1}\eta_1\eta_1^T\eta_2^{-1} + (-2\eta_2)^{-1}\end{aligned}$$

Y se sustituyen los parámetros η_1 y η_2 por los parámetros reales de la distribución *Normal*:

$$\mathbb{E}_{p(x|\mu,\Sigma)}[xx^T] = \mu\mu^T + \Sigma$$

Expectation de los sufficient statistics de la distribución Normal Inverse Wishart

Cálculo de la *expectation* de los *sufficient statistics* parametrizados por η (véase el apéndice A.2.1):

$$\mathbb{E}_{p(\mu,\Sigma|m,\beta,\nu,W)}t(\mu, \Sigma) = \mathbb{E}_{p(\mu,\Sigma|m,\beta,\nu,W)} \begin{bmatrix} \Sigma^{-1}\mu \\ -\frac{1}{2}\Sigma^{-1} \\ -\frac{1}{2}\mu^T\Sigma^{-1}\mu \\ -\frac{1}{2}\ln|\Sigma| \end{bmatrix} = \frac{\partial a(\eta)}{\partial \eta}$$

Expectation del sufficient statistic $\Sigma^{-1}\mu$

$$\begin{aligned}\mathbb{E}_{p(\mu,\Sigma|m,\beta,\nu,W)}[\Sigma^{-1}\mu] &= \frac{\partial a(\eta_1, \eta_2, \eta_3, \eta_4)}{\partial \eta_1} \\ &= -\frac{\eta_4 - D - 2}{2} \cdot \frac{\partial \left(\ln \left| \eta_2 - \frac{1}{\eta_3}\eta_1\eta_1^T \right| \right)}{\partial \eta_1} \\ &= -\frac{\eta_4 - D - 2}{2} \cdot \left(\eta_2 - \frac{1}{\eta_3}\eta_1\eta_1^T \right)^{-1} \left(-\frac{2\eta_1}{\eta_3} \right)\end{aligned}$$

Y se sustituyen los parámetros η_1 , η_2 , η_3 y η_4 por los parámetros reales de la distribución *Normal Inverse Wishart*:

$$\mathbb{E}_{p(\mu,\Sigma|m,\beta,\nu,W)}[\Sigma^{-1}\mu] = m \cdot \nu \cdot W^{-1}$$

Expectation del sufficient statistic $-\frac{1}{2}\Sigma^{-1}$

$$\begin{aligned}\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2}\Sigma^{-1}\right] &= \frac{\partial a(\eta_1, \eta_2, \eta_3, \eta_4)}{\partial \eta_2} \\ &= -\frac{\eta_4 - D - 2}{2} \cdot \frac{\partial \left(\ln \left| \eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right| \right)}{\partial \eta_2} \\ &= -\frac{\eta_4 - D - 2}{2} \cdot \left(\eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right)^{-1}\end{aligned}$$

Y se sustituyen los parámetros η_1 , η_2 , η_3 y η_4 por los parámetros reales de la distribución *Normal Inverse Wishart*:

$$\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2}\Sigma^{-1}\right] = -\frac{1}{2} \cdot \nu \cdot W^{-1}$$

Expectation del sufficient statistic $-\frac{1}{2}\mu^T \Sigma^{-1} \mu$

$$\begin{aligned}\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2}\mu^T \Sigma^{-1} \mu\right] &= \frac{\partial a(\eta_1, \eta_2, \eta_3, \eta_4)}{\partial \eta_3} \\ &= -\frac{\partial \left(\frac{D}{2} \ln |\eta_3| \right)}{\partial \eta_3} - \frac{\partial \left(\frac{\eta_4 - D - 2}{2} \ln \left| \eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right| \right)}{\partial \eta_3} \\ &= -\frac{D}{2\eta_3} - \frac{\eta_4 - D - 2}{2} \cdot \text{Tr} \left(\left(\eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right)^{-1} \left(\frac{1}{\eta_3^2} \eta_1 \eta_1^T \right) \right)\end{aligned}$$

Y se sustituyen los parámetros η_1 , η_2 , η_3 y η_4 por los parámetros reales de la distribución *Normal Inverse Wishart*:

$$\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2}\mu^T \Sigma^{-1} \mu\right] = -\frac{D}{2}\beta^{-1} - \frac{\nu \cdot m^T \cdot W^{-1} \cdot m}{2}$$

Expectation del sufficient statistic $-\frac{1}{2} \ln |\Sigma|$

$$\begin{aligned}\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2} \ln |\Sigma|\right] &= \frac{\partial a(\eta_1, \eta_2, \eta_3, \eta_4)}{\partial \eta_4} \\ &= \frac{\partial}{\partial \eta_4} \left(\frac{(\eta_4 - D - 2)D \cdot \ln 2}{2} \right) + \frac{\partial}{\partial \eta_4} \left(\ln \Gamma_D \left(\frac{\eta_4 - D - 2}{2} \right) \right) \\ &\quad - \frac{\partial}{\partial \eta_4} \left(\frac{\eta_4 - D - 2}{2} \ln \left| \eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right| \right) \\ &= \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi \left(\frac{\eta_4 - D - 2}{2} + \frac{1-i}{2} \right) - \frac{1}{2} \ln \left| \eta_2 - \frac{1}{\eta_3} \eta_1 \eta_1^T \right|\end{aligned}$$

Y se sustituyen los parámetros η_1 , η_2 , η_3 y η_4 por los parámetros reales de la distribución *Normal Inverse Wishart*:

$$\mathbb{E}_{p(\mu, \Sigma|m, \beta, \nu, W)}\left[-\frac{1}{2} \ln |\Sigma|\right] = \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi \left(\frac{\nu}{2} + \frac{1-i}{2} \right) - \frac{1}{2} \ln |W|$$

A.2.3 Formulación del parámetro variacional λ_π en términos de la Exponential Family

En primer lugar hay que definir la distribución *complete conditional* de la variable aleatoria π .

$$p(\pi|c, x, \mu, \Sigma) = p(\pi|c) \text{ ya que } x \perp \pi|c, \mu \perp \pi|c \text{ y } \Sigma \perp \pi|c$$

$$\begin{aligned} p(\pi|c) &\propto p(\pi, c) \\ &\propto p(\pi|\alpha) \prod_{n=1}^N p(c_n|\pi) \end{aligned}$$

Se trata de un submodelo dentro del modelo GMM formado por una distribución *Dirichlet* $p(\pi|\alpha)$ que representa el vector de proporciones de pertenencia de los puntos a cada uno de los *clusters* y N distribuciones *Categorical* $\prod_{n=1}^N p(c_n|\pi)$ que modelan la probabilidad de pertenencia de cada punto a cada *cluster*. Si ahora se formula en base a las propiedades de la *Exponential Family* (véase el apéndice A.2.1) se tiene:

$$\begin{aligned} p(\pi|c) &\propto h(\pi) \cdot \exp(\eta_\pi(\alpha)t(\pi) - a(\eta_\pi(\alpha))) \prod_{n=1}^N h(c_n) \cdot \exp(\eta_{c_n}(\pi)t(c_n) - a(\eta_{c_n}(\pi))) \\ &\propto \exp\left(\begin{bmatrix} \alpha_1 - 1 \\ \vdots \\ \alpha_k - 1 \end{bmatrix}^T \begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_k) \end{bmatrix} - \sum_{k=1}^K \ln(\Gamma(\alpha_k)) + \right. \\ &\quad \left. + \ln\left(\Gamma\left(\sum_{k=1}^K \alpha_k\right)\right) + \sum_{n=1}^N \begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_k) \end{bmatrix}^T \begin{bmatrix} \mathbb{I}(c_n = 1) \\ \vdots \\ \mathbb{I}(c_n = k) \end{bmatrix}\right) \\ &\propto \exp\left(\begin{bmatrix} \alpha_1 - 1 + \sum_{n=1}^N \mathbb{I}(c_n = 1) \\ \vdots \\ \alpha_k - 1 + \sum_{n=1}^N \mathbb{I}(c_n = k) \end{bmatrix}^T \begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_k) \end{bmatrix} - \sum_{k=1}^K \ln(\Gamma(\alpha_k)) + \ln\left(\Gamma\left(\sum_{k=1}^K (\alpha_k)\right)\right)\right) \end{aligned}$$

Como la distribución *Dirichlet* (π) es conjugada de la distribución *Categorical* (c) nos permite sacar factor común en el último paso y la distribución resultando, como se puede ver en su formulación final en *Exponential Family*, es también una distribución *Dirichlet*. A continuación se aproxima la distribución variacional $q(\pi|\lambda_\pi)$ a partir de la distribución *complete conditional* calculada anteriormente mediante la ecuación formulada en A.1.4:

$$q(\pi|\lambda_\pi) \propto \exp(\mathbb{E}_{q(c|\lambda_\phi)} \mathbb{E}_{q(\mu, \Sigma|\lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)} \ln p(\pi|c)) \propto \exp(\mathbb{E}_{q(c|\lambda_\phi)} \ln p(\pi|c))$$

La actualización analítica de λ_π se obtiene igualando los *natural parameters* de la distribución variacional con la *expectation* de los *natural parameters* de la distribución *complete conditional*:

$$\begin{aligned} \eta_{q(i|\lambda_i)} &= \mathbb{E}_{q(\text{sons}(i)|\lambda_{\text{sons}(i)})} \eta_{p(i|\text{sons}(i))} \\ \eta_{q(\pi|\lambda_\pi)} &= \mathbb{E}_{q(c|\lambda_\phi)} \eta_{p(\pi|c)} \end{aligned}$$

Como se ha visto en la derivación de la distribución *complete conditional*, dada la relación de conjugación entre la distribución *Dirichlet* y la distribución *Categorical*, se obtiene una distribución resultante que también es una distribución *Dirichlet*. Si se despejan las λ_{π_i} de la igualdad anterior se obtendrá la actualización analítica del parámetro λ_π :

$$\begin{bmatrix} \lambda_{\pi_1} - 1 \\ \vdots \\ \lambda_{\pi_k} - 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 - 1 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = 1) \\ \vdots \\ \alpha_k - 1 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) \end{bmatrix}$$

A continuación se muestra el resultado de despejar las λ_{π_i} :

$$\lambda_\pi = \begin{bmatrix} \alpha_1 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = 1) \\ \vdots \\ \alpha_k + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) \end{bmatrix}$$

Y se termina sustituyendo las *expectations* por el resultado mostrado en el apéndice A.2.2:

$$\lambda_\pi = \begin{bmatrix} \alpha_1 + \sum_{n=1}^N \lambda_{\phi_{n,1}} \\ \vdots \\ \alpha_k + \sum_{n=1}^N \lambda_{\phi_{n,k}} \end{bmatrix}$$

A.2.4 Formulación de los parámetros variacionales λ_m , λ_β , λ_ν y λ_W en términos de la Exponential Family

En primer lugar hay que definir la distribución *complete conditional* de las variables aleatorias μ y Σ para una k concreta.

$$p(\mu_k, \Sigma_k | c, x, \pi) = p(\mu_k, \Sigma_k | c, x) \text{ ya que } \pi \perp (\mu_k, \Sigma_k) | (c, x)$$

$$\begin{aligned} p(\mu_k, \Sigma_k | c, x) &\propto p(\mu_k, \Sigma_k, c, x) \\ &\propto p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1}^N \prod_{k'=1}^K p(c_n = k' | \pi) p(x_n | \mu_{k'}, \Sigma_{k'})^{\mathbb{I}(c_n = k')} \\ &\propto p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1}^N p(x_n | \mu_k, \Sigma_k)^{\mathbb{I}(c_n = k)} \\ &\propto p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0) \prod_{n=1, c_n=k}^N p(x_n | \mu_k, \Sigma_k) \end{aligned}$$

Se trata de un submodelo dentro del modelo GMM formado por una distribución *Normal Inverse Wishart* $p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0)$ que modela la media y la matriz de covarianzas del *cluster* k y N distribuciones *Normales* $\prod_{n=1, c_n=k}^N p(x_n | \mu_k, \Sigma_k)$ que modelan cada uno de los puntos. Si ahora se formula en base a las propiedades de la *Exponential Family* (véase el apéndice A.2.1) se tiene:

$$\begin{aligned} p(\mu_k, \Sigma_k | c, x) &\propto h(\mu_k, \Sigma_k) \cdot \exp(\eta_{\mu_k, \Sigma_k}(m_0, \beta_0, \nu_0, W_0)t(\mu_k, \Sigma_k) - a(\eta_{\mu_k, \Sigma_k}(m_0, \beta_0, \nu_0, W_0))) \\ &\quad \prod_{n=1, c_n=k}^N h(x_n) \cdot \exp(\eta_{x_n}(\mu_k, \Sigma_k)t(x_n) - a(\eta_{x_n}(\mu_k, \Sigma_k))) \\ &\propto (2\pi)^{-\frac{D}{2}} \cdot \exp\left(\begin{bmatrix} m_0^T \beta_0 \\ W_0 + \beta_0 m_0 m_0^T \\ \beta_0 \\ \nu_0 + D + 2 \end{bmatrix} \begin{bmatrix} \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \Sigma_k^{-1} \\ -\frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \ln |\Sigma_k| \end{bmatrix} - \frac{\nu_0 D}{2} \ln 2 - \ln \Gamma_D\left(\frac{\nu_0}{2}\right) + \frac{D}{2} \ln |\beta_0| + \frac{\nu_0}{2} \ln |W_0|\right) \\ &\quad \cdot (2\pi)^{-\frac{DN_k}{2}} \cdot \exp\left(\begin{bmatrix} \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \Sigma_k^{-1} \end{bmatrix} \begin{bmatrix} \sum_{n=1}^N \mathbb{I}(c_n = k) x_n \\ \sum_{n=1}^N \mathbb{I}(c_n = k) x_n x_n^T \end{bmatrix} - \frac{N_k}{2} \mu_k^T \Sigma_k^{-1} \mu_k - \frac{N_k}{2} \ln |\Sigma_k|\right) \\ &\propto \exp\left(\begin{bmatrix} m_0^T \beta_0 + \sum_{n=1}^N \mathbb{I}(c_n = k) x_n \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \mathbb{I}(c_n = k) x_n x_n^T \\ \beta_0 + N_k \\ \nu_0 + D + 2 + N_k \end{bmatrix} \begin{bmatrix} \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \Sigma_k^{-1} \\ -\frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \ln |\Sigma_k| \end{bmatrix} - \frac{\nu_0 D}{2} \ln 2 - \ln \Gamma_D\left(\frac{\nu_0}{2}\right) + \frac{D}{2} \ln |\beta_0| + \frac{\nu_0}{2} \ln |W_0|\right) \end{aligned}$$

Como la distribución *Normal Inverse Wishart* es conjugada de la distribución *Normal* la probabilidad conjunta de este submodelo también tendrá la forma de una *Normal Inverse Wishart*. A continuación se aproxima la distribución variacional $q(\mu_k, \Sigma_k | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)$ a partir de la distribución *complete conditional* calculada anteriormente mediante la ecuación formulada en A.1.4:

$$\begin{aligned} q(\mu_k, \Sigma_k | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) &\propto \exp(\mathbb{E}_{q(\pi | \lambda_\pi)} \mathbb{E}_{q(c | \lambda_\phi)} \ln p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0)) \\ &\propto \exp(\mathbb{E}_{q(c | \lambda_\phi)} \ln p(\mu_k, \Sigma_k | m_0, \beta_0, \nu_0, W_0)) \end{aligned}$$

La actualización analítica de λ_m , λ_β , λ_ν y λ_W se obtiene igualando los *natural parameters* de la distribución variacional con la *expectation* de los *natural parameters* de la distribución *complete conditional*:

$$\begin{aligned} \eta_{q(i | \lambda_i)} &= \mathbb{E}_{q(\text{sons}(i) | \lambda_{\text{sons}(i)})} \eta_{p(i | \text{sons}(i))} \\ \eta_{q(\mu_k, \Sigma_k | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)} &= \mathbb{E}_{q(c | \lambda_\phi)} \eta_{p(\mu_k, \Sigma_k | c, x)} \end{aligned}$$

Como se ha visto en la derivación de la distribución *complete conditional*, dada la relación de conjugación entre la distribución *Normal Inverse Wishart* y la distribución *Normal*, se obtiene una distribución resultante que

también es una distribución *Normal Inverse Wishart*. Si se despeja λ_m , λ_β , λ_ν y λ_W de la igualdad anterior se obtendrán las actualizaciones analíticas de estos parámetros variacionales:

$$\begin{bmatrix} \lambda_{m_k}^T \lambda_{\beta_k} \\ \lambda_{W_k} + \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \\ \lambda_\beta \\ \lambda_{\nu_k} + D + 2 \end{bmatrix} = \begin{bmatrix} m_0^T \beta_0 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n x_n^T \\ \beta_0 + N_k \\ \nu_0 + D + 2 + N_k \end{bmatrix}$$

A continuación se muestra el resultado de resolver el anterior sistema de ecuaciones y la sustitución de las *expectations* por la solución mostrada en el apéndice A.2.2:

$$\lambda_m = \frac{m_0^T \beta_0 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n}{\lambda_{\beta_k}} = \frac{m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n}{\lambda_{\beta_k}}$$

$$\lambda_\beta = \beta_0 + N_k$$

$$\lambda_\nu = \nu_0 + N_k$$

$$\lambda_W = W_0 + \beta_0 m_0 m_0^T + \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n x_n^T - \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T = W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n x_n^T - \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T$$

Sabiendo que $\mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) = \lambda_{\phi_{n,k}}$ y siendo $N_k = \sum_{n=1}^N \lambda_{\phi_{n,k}}$. Hay que tener en cuenta que cada uno de estos resultados está replicado K veces de la siguiente manera:

$$\lambda_m = \begin{bmatrix} \frac{m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,1}} x_n}{\lambda_{\beta_1}} \\ \dots \\ \frac{m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n}{\lambda_{\beta_k}} \end{bmatrix}$$

$$\lambda_\beta = \begin{bmatrix} \beta_0 + N_1 \\ \dots \\ \beta_0 + N_k \end{bmatrix}$$

$$\lambda_\nu = \begin{bmatrix} \nu_0 + N_1 \\ \dots \\ \nu_0 + N_k \end{bmatrix}$$

$$\lambda_W = \begin{bmatrix} W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,1}} x_n x_n^T - \lambda_{\beta_1} \lambda_{m_1} \lambda_{m_1}^T \\ \dots \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n x_n^T - \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \end{bmatrix}$$

A.2.5 Formulación del parámetro variacional λ_ϕ en términos de la Exponential Family

En primer lugar hay que definir la distribución *complete conditional* de la variable aleatoria c para una n concreta.

$$p(c_n | x_n, \pi, \mu, \Sigma) = p(c_n, x_n | \pi, \mu, \Sigma) \text{ ya que } x_n \not\perp c_n | \pi, \mu, \Sigma$$

$$\begin{aligned} p(c_n, x_n | \pi, \mu, \Sigma) &\propto p(c_n, x_n, \pi, \mu, \Sigma) \\ &\propto p(c_n | \pi) \prod_{k=1}^K p(x_n | \mu_k, \Sigma_k)^{\mathbb{I}(c_n=k)} \end{aligned}$$

Se trata de un submodelo dentro del modelo GMM formado por una distribución *Categorical* $p(c_n | \pi)$ que modela las proporciones de pertenencia del punto n a cada uno de los *clusters* y K distribuciones *Normales* $\prod_{k=1}^K p(x_n | \mu_k, \Sigma_k)^{\mathbb{I}(c_n=k)}$ que modelan la generación del punto n en cada uno de los *clusters*. Si ahora se formula

en base a las propiedades de la *Exponential Family* (véase el apéndice A.2.1) se tiene:

$$\begin{aligned}
p(c_n, x_n | \pi, \mu, \Sigma) &\propto h(c_n) \cdot \exp(\eta_{c_n}(\pi)t(c_n) - a(\eta_{c_n}(\pi))) \prod_{k=1}^K h(x_n) \cdot \exp(\eta_{x_n}(\mu_k, \Sigma_k)t(x_n) - a(\eta_{x_n}(\mu_k, \Sigma_k))) \\
&\propto (2\pi)^{-\frac{DN}{2}} \exp\left(\begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_k) \end{bmatrix}^T \begin{bmatrix} \mathbb{I}(c_n = 1) \\ \vdots \\ \mathbb{I}(c_n = k) \end{bmatrix} + \sum_{k=1}^K \mathbb{I}(c_n = k) \left(\begin{bmatrix} \Sigma_k^{-1} \mu_k \\ -\frac{1}{2} \Sigma_k^{-1} \end{bmatrix}^T \begin{bmatrix} x_n \\ x_n x_n^T \end{bmatrix} - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \ln |\Sigma_k| \right) \right) \\
&\propto (2\pi)^{-\frac{DN}{2}} \exp\left(\left(\begin{bmatrix} \ln(\pi_1) \\ \vdots \\ \ln(\pi_k) \end{bmatrix} + \begin{bmatrix} \Sigma_1^{-1} \mu_1 x_n - \frac{1}{2} \Sigma_1^{-1} x_n x_n^T - \frac{1}{2} \mu_1^T \Sigma_1^{-1} \mu_1 - \frac{1}{2} \ln |\Sigma_1| \\ \vdots \\ \Sigma_k^{-1} \mu_k x_n - \frac{1}{2} \Sigma_k^{-1} x_n x_n^T - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \ln |\Sigma_k| \end{bmatrix} \right) \begin{bmatrix} \mathbb{I}(c_n = 1) \\ \vdots \\ \mathbb{I}(c_n = k) \end{bmatrix} \right)
\end{aligned}$$

A continuación se aproxima la distribución variacional $q(c_n | \lambda_{\phi_n})$ a partir de la distribución *complete conditional* calculada anteriormente mediante la ecuación formulada en A.1.4:

$$q(c_n | \lambda_{\phi_n}) \propto \exp(\mathbb{E}_{q(\pi | \lambda_\pi)} \mathbb{E}_{q(\mu, \Sigma | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)} \ln p(c_n | \pi))$$

La actualización analítica de λ_ϕ se obtiene igualando los *natural parameters* de la distribución variacional con la *expectation* de los *natural parameters* de la distribución *complete conditional*:

$$\begin{aligned}
\eta_{q(i | \lambda_i)} &= \mathbb{E}_{q(\text{sons}(i) | \lambda_{\text{sons}(i)})} \eta_{p(i | \text{sons}(i))} \\
\eta_{q(c_n | \lambda_\phi)} &= \mathbb{E}_{q(\pi | \lambda_\pi)} \mathbb{E}_{q(\mu, \Sigma | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)} \eta_{p(c_n, x_n | \pi, \mu, \Sigma)}
\end{aligned}$$

Si se despeja λ_ϕ de la igualdad anterior se obtendrá la actualización analítica de este parámetro variacional:

$$\begin{bmatrix} \ln \mathbb{E}_{q_c} \mathbb{I}(c_n = 1) \\ \vdots \\ \ln \mathbb{E}_{q_c} \mathbb{I}(c_n = k) \end{bmatrix} = \begin{bmatrix} \mathbb{E}_{q_\pi} \ln \pi_1 + \mathbb{E}_{q_{\mu, \Sigma}} \Sigma_1^{-1} \mu_1 x_n + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \Sigma_1^{-1} x_n x_n^T + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \mu_1^T \Sigma_1^{-1} \mu_1 + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \ln |\Sigma_1| \\ \vdots \\ \mathbb{E}_{q_\pi} \ln \pi_k + \mathbb{E}_{q_{\mu, \Sigma}} \Sigma_k^{-1} \mu_k x_n + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \Sigma_k^{-1} x_n x_n^T + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k + \mathbb{E}_{q_{\mu, \Sigma}} - \frac{1}{2} \ln |\Sigma_k| \end{bmatrix}$$

Siendo $q_c = q(c | \lambda_\phi)$, $q_\pi = q(\pi | \lambda_\pi)$ y $q_{\mu, \Sigma} = q(\mu, \Sigma | \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)$. A continuación se muestra el resultado de resolver el anterior sistema de ecuaciones y la sustitución de las *expectations* por la solución mostrada en los apéndices A.2.2, A.2.2 y A.2.2:

$$\lambda_\phi = \text{Softmax} \left(\begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \lambda_{m_1} \lambda_{\nu_1} \lambda_{W_1}^{-1} x_n - \frac{1}{2} \lambda_{\nu_1} \lambda_{W_1}^{-1} x_n x_n^T \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \lambda_{m_k} \lambda_{\nu_k} \lambda_{W_k}^{-1} x_n - \frac{1}{2} \lambda_{\nu_k} \lambda_{W_k}^{-1} x_n x_n^T \\ -\frac{D}{2} \lambda_{\beta_1}^{-1} - \frac{\lambda_{\nu_1} \lambda_{m_1}^T \lambda_{W_1}^{-1} \lambda_{m_1}}{2} + \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_1}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_1}| \\ \vdots \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \lambda_{m_k}^T \lambda_{W_k}^{-1} \lambda_{m_k}}{2} + \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} \right)$$

$$\text{Donde } \text{Softmax} \left(\begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} \right) = \begin{bmatrix} \frac{\exp(x_1)}{\sum_{k=1}^K x_k} \\ \vdots \\ \frac{\exp(x_k)}{\sum_{k=1}^K x_k} \end{bmatrix}$$

A.2.6 Derivación completa de la ELBO

La derivación empieza a partir de la definición de la ELBO a la que se ha llegado anteriormente:

$$ELBO(q(\theta|\lambda), p(x, \theta)) = \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$$

Que en términos del modelo GMM queda como:

$$ELBO(q(\theta|\lambda), p(x, \theta)) = \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta|\alpha, m_0, \beta_0, \nu_0, W_0)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$$

siendo $\theta = [\pi, c, \mu_k, \Sigma_k]$ y $\lambda = [\lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W]$.

Seguidamente se muestra la derivación de $\mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta|\alpha, m_0, \beta_0, \nu_0, W_0))$ y $\mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda))$ por separado.

Derivación de $\mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \pi, c, \mu, \Sigma|\alpha, m_0, \beta_0, \nu_0, W_0))$

$$\begin{aligned} & \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \pi, c, \mu, \Sigma|\alpha, m_0, \beta_0, \nu_0, W_0)) = \\ &= \mathbb{E}_{q(\theta|\lambda)} \left(\ln p(\pi|\alpha) + \sum_{n=1}^N \ln p(c_n|\pi) + \sum_{k=1}^K \left(\ln p(\mu_k, \Sigma_k|m_0, \beta_0, \nu_0, W_0) + \sum_{n=1, \mathbb{I}(c_n=k)}^N \ln p(x_n|\mu_k, \Sigma_k) \right) \right) \\ &= \begin{bmatrix} \alpha_1 - 1 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = 1) \\ \vdots \\ \alpha_k - 1 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) \end{bmatrix}^T \begin{bmatrix} \mathbb{E}_{q(\pi|\lambda_\pi)} \ln(\pi_1) \\ \vdots \\ \mathbb{E}_{q(\pi|\lambda_\pi)} \ln(\pi_k) \end{bmatrix} - \sum_{i=1}^K \ln \Gamma(\alpha_i) + \ln \Gamma \left(\sum_{j=1}^K \alpha_j \right) - \frac{D(N+1)}{2} K \cdot \ln(2\pi) \\ &+ \sum_{k=1}^K \begin{bmatrix} m_0^T \beta_0 + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) x_n x_n^T \\ \beta_0 + N_k \\ \nu_0 + D + 2 + N_k \end{bmatrix}^T \begin{bmatrix} \mathbb{E}_{q(\mu_m|\lambda_m)} \Sigma_k^{-1} \mu_k \\ \mathbb{E}_{q(\mu_\beta|\lambda_\beta)} - \frac{1}{2} \Sigma_k^{-1} \\ \mathbb{E}_{q(\Sigma_\nu|\lambda_\nu)} - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k \\ \mathbb{E}_{q(\Sigma_W|\lambda_W)} - \frac{1}{2} \ln |\Sigma_k| \end{bmatrix} \\ &- K \cdot \frac{\nu_0 D}{2} \ln 2 - K \cdot \ln \Gamma_D \left(\frac{\nu_0}{2} \right) + K \cdot \frac{D}{2} \ln |\beta_0| + K \cdot \frac{\nu_0}{2} \ln |W_0| \end{aligned}$$

Y se sustituyen las *expectations* por las soluciones calculadas en el apéndice A.2.2:

$$\begin{aligned} & \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \pi, c, \mu, \Sigma|\alpha, m_0, \beta_0, \nu_0, W_0)) = \\ &= \begin{bmatrix} \alpha_1 - 1 + \sum_{n=1}^N \lambda_{\phi_{n,1}} \\ \vdots \\ \alpha_k - 1 + \sum_{n=1}^N \lambda_{\phi_{n,k}} \end{bmatrix}^T \begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi \left(\sum_{i=1}^K \lambda_{\pi_i} \right) \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi \left(\sum_{i=1}^K \lambda_{\pi_i} \right) \end{bmatrix} - \sum_{i=1}^K \ln \Gamma(\alpha_i) + \ln \Gamma \left(\sum_{j=1}^K \alpha_j \right) - \frac{D(N+1)}{2} K \cdot \ln(2\pi) \\ &+ \sum_{k=1}^K \begin{bmatrix} m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n x_n^T \\ \beta_0 + N_k \\ \nu_0 + D + 2 + N_k \end{bmatrix}^T \begin{bmatrix} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k} \\ -\frac{1}{2} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \cdot \lambda_{m_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k}}{2} \\ \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi \left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2} \right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} \\ &- K \cdot \frac{\nu_0 D}{2} \ln 2 - K \cdot \ln \Gamma_D \left(\frac{\nu_0}{2} \right) + K \cdot \frac{D}{2} \ln |\beta_0| + K \cdot \frac{\nu_0}{2} \ln |W_0| \end{aligned}$$

Derivación de $\mathbb{E}_{q(\theta|\lambda)}(\ln q(\pi, c, \mu, \Sigma|\lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W))$

$$\begin{aligned}
& \mathbb{E}_{q(\theta|\lambda)} (\ln q(\pi, c, \mu, \Sigma|\lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)) = \\
& = \mathbb{E}_{q(\theta|\lambda)} \left(\ln q(\pi|\lambda_\pi) + \sum_{k=1}^K \ln q(\mu_k, \Sigma_k|\lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W) + \sum_{n=1}^N \ln q(c_n|\lambda_\phi) \right) \\
& = \mathbb{E}_{q(\theta|\lambda)} \left(\eta(\lambda_\pi)t(\pi) - a(\eta(\lambda_\pi)) + \sum_{k=1}^K \ln h(\mu_k, \Sigma_k) + \eta(\lambda_{m_k}, \lambda_{\beta_k}, \lambda_{\nu_k}, \lambda_{W_k})t(\mu_k, \Sigma_k) \right. \\
& \quad \left. - a(\eta(\lambda_{m_k}, \lambda_{\beta_k}, \lambda_{\nu_k}, \lambda_{W_k})) \right) + \mathbb{E}_{q(\theta|\lambda)} \left(\sum_{n=1}^N \eta(\lambda_\phi t(c_n)) \right) \\
& = \begin{bmatrix} \lambda_{\pi_1} - 1 \\ \dots \\ \lambda_{\pi_K} - 1 \end{bmatrix}^T \begin{bmatrix} \mathbb{E}_{q(\pi|\lambda_\pi)} \ln \pi_1 \\ \dots \\ \mathbb{E}_{q(\pi|\lambda_\pi)} \ln \pi_K \end{bmatrix} - \sum_{i=1}^K \ln \Gamma(\lambda_{\pi_i}) + \ln \Gamma\left(\sum_{i=1}^K \lambda_{\pi_i}\right) - \frac{D}{2} K \cdot \ln(2\pi) \\
& \quad + \sum_{k=1}^K \begin{bmatrix} \lambda_{m_k}^T \lambda_{\beta_k} \\ \lambda_{W_k} + \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \\ \lambda_{\beta_k} \\ \lambda_{\nu_k} + D + 2 \end{bmatrix}^T \begin{bmatrix} \mathbb{E}_{q(\mu_m|\lambda_m)} \Sigma_k^{-1} \mu_k \\ \mathbb{E}_{q(\mu_\beta|\lambda_\beta)} - \frac{1}{2} \Sigma_k^{-1} \\ \mathbb{E}_{q(\Sigma_\nu|\lambda_\nu)} - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k \\ \mathbb{E}_{q(\Sigma_W|\lambda_W)} - \frac{1}{2} \ln |\Sigma_k| \end{bmatrix} - \frac{\lambda_{\nu_k} D}{2} \ln 2 - \ln \Gamma_D\left(\frac{\lambda_{\nu_k}}{2}\right) \\
& \quad + \frac{D}{2} \ln |\lambda_{\beta_k}| + \frac{\lambda_{\nu_k}}{2} \ln |\lambda_{W_k}| + \sum_{n=1}^N \begin{bmatrix} \ln \mathbb{E}_{q(\theta|\lambda)} \mathbb{I}(c_n = 1) \\ \dots \\ \ln \mathbb{E}_{q(\theta|\lambda)} \mathbb{I}(c_n = k) \end{bmatrix} \begin{bmatrix} \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = 1) \\ \dots \\ \mathbb{E}_{q(c|\lambda_\phi)} \mathbb{I}(c_n = k) \end{bmatrix}
\end{aligned}$$

Y se sustituyen las *expectations* por las soluciones calculadas en el apéndice A.2.2:

$$\begin{aligned}
& \mathbb{E}_{q(\theta|\lambda)} (\ln q(\pi, c, \mu, \Sigma|\lambda_\pi, \lambda_\phi, \lambda_m, \lambda_\beta, \lambda_\nu, \lambda_W)) = \\
& = \begin{bmatrix} \lambda_{\pi_1} - 1 \\ \dots \\ \lambda_{\pi_K} - 1 \end{bmatrix}^T \begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \\ \dots \\ \Psi(\lambda_{\pi_K}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \end{bmatrix} - \sum_{i=1}^K \ln \Gamma(\lambda_{\pi_i}) + \ln \Gamma\left(\sum_{i=1}^K \lambda_{\pi_i}\right) - \frac{D}{2} K \cdot \ln(2\pi) \\
& \quad + \sum_{k=1}^K \begin{bmatrix} \lambda_{m_k}^T \lambda_{\beta_k} \\ \lambda_{W_k} + \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \\ \lambda_{\beta_k} \\ \lambda_{\nu_k} + D + 2 \end{bmatrix}^T \begin{bmatrix} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k} \\ -\frac{1}{2} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \cdot \lambda_{m_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k}}{2} \\ \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} - \frac{\lambda_{\nu_k} D}{2} \ln 2 - \ln \Gamma_D\left(\frac{\lambda_{\nu_k}}{2}\right) \\
& \quad + \frac{D}{2} \ln |\lambda_{\beta_k}| + \frac{\lambda_{\nu_k}}{2} \ln |\lambda_{W_k}| + \sum_{n=1}^N \begin{bmatrix} \ln \lambda_{\phi_{n,1}} \\ \dots \\ \ln \lambda_{\phi_{n,k}} \end{bmatrix} \begin{bmatrix} \lambda_{\phi_{n,1}} \\ \dots \\ \lambda_{\phi_{n,k}} \end{bmatrix}
\end{aligned}$$

Resultado final

$$\begin{aligned}
ELBO(q(\theta|\lambda), p(x, \theta)) &= \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda)) = \\
&= \begin{bmatrix} \alpha_1 - 1 + \sum_{n=1}^N \lambda_{\phi_{n,1}} \\ \vdots \\ \alpha_k - 1 + \sum_{n=1}^N \lambda_{\phi_{n,k}} \end{bmatrix}^T \begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \end{bmatrix} - \sum_{i=1}^K \ln \Gamma(\alpha_i) + \ln \Gamma\left(\sum_{j=1}^K \alpha_j\right) - \frac{D(N+1)}{2} K \cdot \ln(2\pi) \\
&\quad + \sum_{k=1}^K \begin{bmatrix} m_0^T \beta_0 + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n \\ W_0 + \beta_0 m_0 m_0^T + \sum_{n=1}^N \lambda_{\phi_{n,k}} x_n x_n^T \\ \beta_0 + N_k \\ \nu_0 + D + 2 + N_k \end{bmatrix}^T \begin{bmatrix} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k} \\ -\frac{1}{2} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \cdot \lambda_{m_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k}}{2} \\ \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} \\
&\quad - K \cdot \frac{\nu_0 D}{2} \ln 2 - K \cdot \ln \Gamma_D\left(\frac{\nu_0}{2}\right) + K \cdot \frac{D}{2} \ln |\beta_0| + K \cdot \frac{\nu_0}{2} \ln |W_0| \\
&\quad - \begin{bmatrix} \lambda_{\pi_1} - 1 \\ \vdots \\ \lambda_{\pi_k} - 1 \end{bmatrix}^T \begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \end{bmatrix} + \sum_{i=1}^K \ln \Gamma(\lambda_{\pi_i}) - \ln \Gamma\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \frac{D}{2} K \cdot \ln(2\pi) \\
&\quad - \sum_{k=1}^K \begin{bmatrix} \lambda_{m_k}^T \lambda_{\beta_k} \\ \lambda_{W_k} + \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \\ \lambda_{\beta_k} \\ \lambda_{\nu_k} + D + 2 \end{bmatrix}^T \begin{bmatrix} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k} \\ -\frac{1}{2} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \cdot \lambda_{m_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k}}{2} \\ \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix} + \frac{\lambda_{\nu_k} D}{2} \ln 2 + \ln \Gamma_D\left(\frac{\lambda_{\nu_k}}{2}\right) \\
&\quad - \frac{D}{2} \ln |\lambda_{\beta_k}| - \frac{\lambda_{\nu_k}}{2} \ln |\lambda_{W_k}| - \sum_{n=1}^N \begin{bmatrix} \ln \lambda_{\phi_{n,1}} \\ \vdots \\ \ln \lambda_{\phi_{n,k}} \end{bmatrix} \begin{bmatrix} \lambda_{\phi_{n,1}} \\ \vdots \\ \lambda_{\phi_{n,k}} \end{bmatrix}
\end{aligned}$$

Si ahora se saca factor común $\begin{bmatrix} \Psi(\lambda_{\pi_1}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \\ \vdots \\ \Psi(\lambda_{\pi_k}) - \Psi\left(\sum_{i=1}^K \lambda_{\pi_i}\right) \end{bmatrix}$ y $\begin{bmatrix} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k} \\ -\frac{1}{2} \lambda_{\nu_k} \cdot \lambda_{W_k}^{-1} \\ -\frac{D}{2} \lambda_{\beta_k}^{-1} - \frac{\lambda_{\nu_k} \cdot \lambda_{m_k} \cdot \lambda_{W_k}^{-1} \cdot \lambda_{m_k}}{2} \\ \frac{D}{2} \ln 2 + \frac{1}{2} \sum_{i=1}^D \Psi\left(\frac{\lambda_{\nu_k}}{2} + \frac{1-i}{2}\right) - \frac{1}{2} \ln |\lambda_{W_k}| \end{bmatrix}$, se

sustituyen las λ_{π_s} de la expresión $\begin{bmatrix} \lambda_{\pi_1} - 1 \\ \vdots \\ \lambda_{\pi_k} - 1 \end{bmatrix}$ por la expresión obtenida en A.2.3 y se sustituyen λ_m , λ_β , λ_ν y

λ_W de $\begin{bmatrix} \lambda_{m_k}^T \lambda_{\beta_k} \\ \lambda_{W_k} + \lambda_{\beta_k} \lambda_{m_k} \lambda_{m_k}^T \\ \lambda_{\beta_k} \\ \lambda_{\nu_k} + D + 2 \end{bmatrix}$ por las expresiones obtenidas en A.2.4 se obtiene una versión simplificada.

$$\begin{aligned}
ELBO(q(\theta|\lambda), p(x, \theta)) &= \mathbb{E}_{q(\theta|\lambda)}(\ln p(x, \theta)) - \mathbb{E}_{q(\theta|\lambda)}(\ln q(\theta|\lambda)) = \\
&= - \sum_{i=1}^K \ln \Gamma(\alpha_i) + \ln \Gamma\left(\sum_{j=1}^K \alpha_j\right) - \frac{D(N+1)}{2} K \cdot \ln(2\pi) - K \cdot \frac{\nu_0 D}{2} \ln 2 - K \cdot \ln \Gamma_D\left(\frac{\nu_0}{2}\right) \\
&\quad + K \cdot \frac{D}{2} \ln |\beta_0| + K \cdot \frac{\nu_0}{2} \ln |W_0| + \sum_{i=1}^K \ln \Gamma(\lambda_{\pi_i}) - \ln \Gamma\left(\sum_{i=1}^K \lambda_{\pi_i}\right) + \frac{D}{2} K \cdot \ln(2\pi) + \frac{\lambda_{\nu_k} D}{2} \ln 2 \\
&\quad + \ln \Gamma_D\left(\frac{\lambda_{\nu_k}}{2}\right) - \frac{D}{2} \ln |\lambda_{\beta_k}| - \frac{\lambda_{\nu_k}}{2} \ln |\lambda_{W_k}| - \sum_{n=1}^N \begin{bmatrix} \ln \lambda_{\phi_{n,1}} \\ \vdots \\ \ln \lambda_{\phi_{n,k}} \end{bmatrix} \begin{bmatrix} \lambda_{\phi_{n,1}} \\ \vdots \\ \lambda_{\phi_{n,k}} \end{bmatrix}
\end{aligned}$$

De esta última expresión es de la que se ha realizado la implementación para el algoritmo de *Coordinate Ascent Variational Inference* (CAVI)¹.

¹Código: https://github.com/bertini36/GMM/blob/master/inference/python/gmm_cavi.py

Referencias

- [1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow.org*.
- [2] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.
- [3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.
- [4] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.
- [5] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 936. John Wiley & Sons, 2012.
- [6] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [7] J Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [8] John Burkardt. K-means clustering. *Virginia Tech, Advanced Research Computing, Interdisciplinary Center for Applied Mathematics*, 2009.
- [9] Rendi Pratama. Review of density-based spatial clustering algorithm.
- [10] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, volume 84. Marcel Dekker, 1988.
- [11] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [12] Christopher M Bishop. Model-based machine learning. *Phil. Trans. R. Soc. A*, 371(1984):20120222, 2013.
- [13] Jian-Xin Pan and Kai-Tai Fang. Maximum likelihood estimation. *Growth Curve Models and Statistical Diagnostics*, pages 77–158, 2002.
- [14] J-L Gauvain and Chin-Hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE transactions on speech and audio processing*, 2(2):291–298, 1994.
- [15] George EP Box and George C Tiao. *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons, 2011.
- [16] Arian Maleki and Tom Do. Review of probability theory. *CS*, 229(2):1.
- [17] George E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.
- [18] David M. Blei. Build, compute, critique, repeat: Data analysis with latent variable models. 2016.
- [19] K Binder. Ising model. hazewinkel, michiel, encyclopedia of mathematics, 2001.
- [20] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.
- [21] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.
- [22] David M. Blei. Variational inference.
- [23] Ilker Yildirim. Bayesian inference: Gibbs sampling. *Technical Note, University of Rochester*, 2012.

- [24] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.
- [25] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.
- [26] John Winn and Christopher M Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.
- [27] Rajesh Ranganath, Dustin Tran, Jaan Altosaar, and David Blei. Operator variational inference. In *Advances in Neural Information Processing Systems*, pages 496–504, 2016.
- [28] John H Halton. Sequential monte carlo. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 58, pages 57–78. Cambridge Univ Press, 1962.
- [29] Sherman Ip and Jack Jewson. Hamiltonian monte carlo. 2015.
- [30] Dustin Tran, Alp Kucukelbir, Adji B Dieng, Maja Rudolph, Dawen Liang, and David M Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.
- [31] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.
- [32] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *arXiv preprint arXiv:1603.00788*, 2016.
- [33] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.
- [34] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.
- [35] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [36] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.
- [37] Variational bayesian inference (slides).
- [38] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- [39] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [40] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [41] John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.
- [42] Dustin Tran, Panos Toulis, and Edoardo M Airoldi. Stochastic gradient descent methods for estimation with large data sets. *arXiv preprint arXiv:1509.06459*, 2015.
- [43] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
- [44] Francisco R Ruiz, Michalis Titsias RC AUEB, and David Blei. The generalized reparameterization gradient. In *Advances in Neural Information Processing Systems*, pages 460–468, 2016.
- [45] Eric W Weisstein. Monte carlo integration. 2003.
- [46] Bob Carpenter, Matthew D Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The stan math library: Reverse-mode automatic differentiation in c++. *arXiv preprint arXiv:1509.07164*, 2015.
- [47] Joan Capdevila, Jesús Cerquides, and Jordi Torres. Mining urban events in the tweet stream through a probabilistic mixture model. pages x–x, 2017.
- [48] Joan Capdevila, Jesús Cerquides, and Jordi Torres. Recognizing warblers: a probabilistic model for event detection in twitter. 2016.

- [49] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.
- [50] CUDA Nvidia. Compute unified device architecture programming guide. 2007.
- [51] Víctor Campos, Francesc Sastre, Maurici Yagües, Míriam Bellver, X. Giró-i Nieto, and Jordi Torres. Distributed training strategies for a computer vision deep learning algorithm on a distributed gpu cluster. In *International Conference on Computational Science (ICCS)*, Zurich, Switzerland, 06/2017 In Press.
- [52] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
- [53] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [54] Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.
- [55] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.
- [56] Matthew Johnson, David K Duvenaud, Alex Wiltchko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, pages 2946–2954, 2016.

